On Affinity Based Routing in Multi-System Data Sharing

Philip S. Yu, Douglas W. Cornell, Daniel M. Dias and Balakrishna R. Iyer

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

Abstract

Multiple systems coupling incurs performance degradation due to inter-system (global) lock contention and database buffer invalidation. At high transaction rates, the level of inter-system interference can have a severe impact on performance. In this paper, we propose a scheme for transaction routing that reduces inter-system interference while keeping load nearly balanced. The routing decision is based on affinity relations defined between transactions and databases. A methodology, employing an integer linear programming technique, is developed to classify incoming transactions into affinity groups based on their *database call reference pattern*. Based on traces from two of IBM's high volume single system customers, we find that, at high transaction rates, the proposed affinity based routing significantly reduces the lock contention probability and leads to a substantial reduction in transaction response time. Further, the reduction in inter-system data contention, produces a large impact on the performance of optimistic type concurrency control.

1. Introduction

The rapid increase in MIPS required for transaction processing and the inability of high-end processor MIPS to keep up with demand have required the design of multiple processor based architectures. In a multi-system data sharing environment [SEKI84], processors running independent copies of the operating system are coupled by sharing the database on the disks. Every system has direct access to the shared database. In [YU85A,B], a comprehensive study is provided to understand the performance impact of database system users migrating from a single system to a multi-system environment. The two sources of intersystem interference in IBM's IMS database managerment system [STRI82] are: (a) lock contention due to transaction on different systems needing to simultaneously hold the right to access the same data, and (b) database buffer invalidation due to updates of a block on one system causing obsolescence of copies of the same block if retained by other systems. The level of interference among systems, especially due to lock contention, is found to have a critical effect on system performance, as the transaction rate increases.

In a multi-system transaction processing environment, a common front-end processor may be used to route incoming transactions from terminals to transaction processors. Inter-system interference may be reduced by employing a proper routing strategy at the front-end. Intuitively, if transactions with similar database reference patterns are routed to the same system, the level of interference (lock conflicts and buffer invalidations) may be reduced. This is referred to as affinity based routing [SHOE84]. The question is how to classify transactions into affinity groups and assign affinity groups to systems, while keeping the system load balanced. In this paper, we develop a methodology to partition transactions into affinity groups. We evaluate the effectiveness of affinity based routing on reducing inter-system interference. Furthermore, the effect of affinity based routing on an optimistic type of concurrency control is investigated [KUNG81] [SHOE84]. Our investigations are based on workloads traced on two mainframe systems, each running a commercial high volume database system. Although our conclusions may be only strictly valid for the two traced workloads, they show clear trends that we conjecture to be characteristics of many more workloads. The methodology developed is applicable to all workloads.

To study response time under different system structures and configurations, e.g. processor size, concurrency control schemes, etc., a hierarchical modelling approach is taken to decompose the problem into manageable parts and gain the benefit of both trace driven simulation and analytical modelling. We decompose the problem into two parts. In the first part, a trace driven simulation is employed to study the effect of the workload on lock contention and buffer invalidation. The lock contention observed is based on the original timing of the lock request trace. In the second part, an approximate analytical queueing model is adopted to study the effect of system structures and configurations on response time. The analytical model uses the lock contention estimate from the trace driven simulation as an initial estimate of the lock contention probability. A methodology is developed to adjust this initial estimate taking into account the effect of global locking overhead, communication delay based on the system structure and configuration. The analytical model assumes a linear relationship between lock contention and transaction response time to adaptively adjust the lock contention probability.

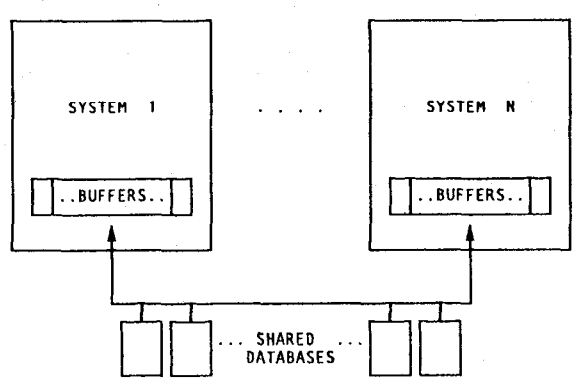

Figure 1.1. Multisystem Data Sharing

Proceedings of the Twelfth International
Conference on Very Large Data Bases

Kyoto, August, 1986

An overview of the techniques used in IMS multi-system data sharing is given in Section 2. In Section 3, we present a routing scheme to exploit transaction affinity. In order to evaluate the reduction in inter-system interference due lo routing by affinity, as compared to the case without routing, multi-system lock traces are synthesized from traces obtained from two users of IBM's IMS database system. Trace driven simulations are employed. The methodology is discussed in Section 4. In Section 5, the benefits of transaction routing by affinity are examined. Both the impact on lock contention, and on buffer invalidation and I/O rate are considered. A study of overall system performance is conducted in Section 6. The effect of routing on a semi-optimistic concurrency control scheme is also considered. Concluding remarks appear in Section 7. Affinity routing is found to significantly reduce transaction response time at high transaction rates.

2. Overview of IMS Data Sharing

The analysis in this paper is based on traces from IBM's IMS database management system [DATE86]. This section outlines aspects of IMS pertinent to following sections. Each system schedules transactions for execution in a fixed number of *regions* (a region can be thought of as a single process that executes transactions sequentially). Between systems, read and write locks are obtained at record granularity. Although records can span blocks in some access methods, typically there are multiple records stored in each block. Additionally, for each block containing an update, an exclusive lock is obtained for the block -- this is necessary to prevent a record that was updated by one system being overwritten with its old value by a concurrent update to a different record in the same block by the second system. Note that although this is called block level data sharing, if one system is updating a record, concurrent reads to different records in the same block by the second system are allowed. More precisely, data sharing is provided at the record level for read access, and at the block level for updates. Finally, dataset busy locks are obtained for various operations such as update or insertion into a database index. The locking protocol, both within a system and between systems, is not strictly lwo-phase [ESWA76], but rather depends on the semantics of the database operations being performed. We will call the locks obtained for concurrency control within a single system (intra-system) *local locks*, and the inter-system locks *global locks*.

The IMS resource lock manager (JRLM) implements global locking through a protocol called ***pass-the-buck***. The two systems exchange messages to pass a "buck" (table) back and forth. The buck is alternately held by each system for a fixed time called the buck hold time and is then sent to the other system, incurring a communication delay. The sum of the buck hold time and the buck communication delay will be referred to as the ***buck delay***. On each system, global lock requests are generally queued in the intervals between the points at which the buck is sent to the other system. Sending the buck lo the other system consists of sending all queued global lock requests, together with buffer invalidation messages (see below). When a system receives the buck, it processes the global lock requests and invalidation messages from the other system, and then sends back the results of these requests together with its own queue of requests when the buck hold time expires. Generally, pass-the-buck protocol requires each global lock request to be granted by both systems. There are exceptions to this due to the use of a lock name hashing scheme, which we describe next.

In order to reduce communication overhead and the average response time for a global lock request, a lock name hashing scheme is used by IMS: every lock name hashes to one of 16K hash classes. IMS uses a global hash table to track the states of each hash class. For each hash class, a hash table entry contains an *interest bit*, for each system, with the following semantics: a system has interest in a hash class if it currently owns or is waiting for any locks in that hash class. We will use the following notation (different from that of IRLM) for the hash class stales.

[0,0] no syslem has interest in this hash class;
[1,0] only this system has interest locks in this hash class;
[0,1] some other system has interest in this hash class;
[1,1] both this and other system(s) have interest in this hash class.

Any system can immediately grant a lock that maps to a hash class in state [1,0] at that system. If a lock maps to a hash class in state [0,0], the lock can be granted when the buck is sent and acknowledged by the next system in the order in which the buck is passed, but it is not necessary to wait for the buck to return before granting the lock. For other states, the requesting system must send out the buck and wait until the buck returns and it is determined that there is no contention on the specific lock requested.

Finally, messages to invalidate buffers are included in lhe buck in the pass-the-buck protocol, and any blocks referred to in the invalidate messages that happen to reside in buffers on the system receiving the buck are marked as invalid. When a transaction completes, its update locks are held at least until the buck is sent to the other system and the invalidate processing is acknowledged as complete, so that it is known that any old versions of the blocks being updated have been invalidated (update locks are also held until log buffers and updated blocks have been forced to disk).

3. Transaction Routing by Affinity

The problem of routing transactions among multiple systems in order to optimize overall system performance involves minimizing inter-system interference while simultaneously balancing the load on the systems in the complex. The routing scheme considered here is a static routing scheme where all transactions of a type will be routed to a fixed destination system. We assume transaction types are known apriori, e.g. in transactions generated by pre-coded application programs by supplying different parameters.

Trace tapes of database requests are used to establish the viability of affinity based transaction routing. Two large high volume single system IMS installations were traced. The first system ran a parts database system on an IBM 3081K, with 158 physical databases. The tracing period was about 15 minutes. The second system ran on on-line planning database system on an IBM 3033, wilh 103 physical databases. The tracing period was about GO minutes. (In IMS, a physical database is an ordered set of physical database records which in turn are physical sets of hierarchically-related segments of one or more segment types.) The two traces will be referred to as workloads 1 and 2, hereafter.

Each trace tape has records including the identifier of the *re*source to be locked, transaction name and time. Transactions generated by the same application program have the same name. The transaction name will be the basis of classification of transactions into types. The resource identifier is a concatenation of database number, dataset number, and offset within dataset which is referred to as the relative byte address (RBA) of the resource. A data block reference trace can be derived by converting the RBA of a record or segment to be locked into the RBA of the block containing it. A trace processing program was developed to measure the number of occurrences of differenl data block references, read and write, to each database, etc, for each transaction type.

We will try to assign transactions referenci g common databases to the same system. A database is considered to be affiliated to the system with the most references to it. We hypothesize that minimizing database references from transactions assigned to a system to databases affiliated with other systems, will reduce inter-system interference. This hypothesis is verified later.

Let $x(i,j)$ be the transaction routing matrix and $y(i,j)$ be the database affinity matrix. Define $x(i\,j) = 1$ if transaction type i is assigned to affinity group j and $x(i,j) = 0$ otherwise. Similarly, $y(i,j) = 1$ if database i belongs to affinity group j and $y(ij) = 0$ otherwise. If i is a transaction and k is a database, the penalty for $x(i,j)y(k,m)$ having a value 1 is the number of data block reference calls from transaction type i *to* database k if $j \neq m$ and is 0 if $j = m$.

$$\text{Penalty} = \sum_i \sum_j \sum_k \sum_{\substack{m \\ j \neq m}} a(i,k)\, x(i,j)\, y(k,m) \ ,$$

where $a(i,k)$ is the number of different data block references from transaction type i to database k and can be obtained from the trace processing program. The objective is to minimize the penalty subject to the constraints,

$$\sum_j x(i,j) = 1 \ \forall i, \quad \text{and} \quad \sum_m y(k,m) = 1 \ \forall k \ .$$

This quadratic programming problem can be transformed into a linear programming problem in integer variables by a standard method [WATE67]. An additional set of constraints on the solution is used to balance the load. Let $D(i)$ be the number of database calls made by a transaction of type i and $\lambda(i)$ be the arrival rate of transaction type i. Note that $D(i)$ is independent of the assignment of transaction types or databases to systems. Assume that α is the average number of instructions on a transaction application program, and β is average number of instructions to process a database call. The constraint on the load to system j is as follows:

$$\text{lowerlimit} \ \leq \ \text{workload}(j) \ \leq \ \text{upperlimit}$$

where

$$\text{workload}(j) = \sum_i \lambda(i)(\alpha x(i,j) + \beta D(i) x(i,j))$$

All $\lambda(i)$ and $D(i)$ can be derived from the trace processing. The upperlimit and lowerlimit values are chosen to keep each system's load to within 10% of the average system load. The parameters α and β have the values 100K and 15K, respectively. In setting up the linear integer programming it is not necessary to include all of the transaction types and databases. Good results can be obtained by performing clustering on the thirty or so most active transactions types and databases using the linear integer programming approach, where rest of the transactions can be assigned to systems based on heuristics.

4. Multi-system Trace Synthesis and Trace Driven Simulation

To evaluate the benefit from the affinity based routing strategy, trace driven simulations are employed to study the inter-system interference statistics. In the first case (no routing) every transaction type can be executed on any system. In the second case, routing is employed based on transaction affinity, as described in Section 3.

To conduct the trace driven simulations, multi-system lock traces and reference traces are synthesized from single system IMS traces for each of the two cases, respectively. These characterized the type of workloads we would have, if a single system IMS workload was required to move to a multi-system data sharing environment due to increased throughput requirements, assuming however that application programs and databases remain essentially unchanged. A methodology to synthesize a multi-system lock trace for the case without routing is described in [YU85B]. Inter-system or global lock requests are derived from the single system trace by analyzing the semantics of the particular lock requests. The trace tape is split by time interval. In a two system example, the lock records in each region for the second half of the trace period are moved "backward in time" to the beginning of the trace period and assumed to occur from the second system.

Next consider the multi-system lock trace for the case of affinity based routing. To make the two cases comparable, we take the multi-system traces for no routing and remap it into another multi-system trace which captures the characteristic of routing by affinity. The same level of multiprogramming or number of regions is maintained. The order of transaction arrivals to the front-end processor is preserved from the case with no routing. Transactions are reassigned to systems based on the integer programming solution of the problem to reduce inter-system interference as described in Section 3. The multi-system trace from the no routing case is first sorted by transaction start time. A scheduling routine is developed to scan the sorted trace and schedule the transaction assigned to each system in an appropriate region. The transactions cannot remain assigned to the same region as in the original single system trace. For each system, the transactions are reassigned to regions so as to approximately equalize the run times in each region. The scheduling routine does not exploit any sophisticated technique to minimize the local lock contention. Some attempt is made to reduce local lock contention by attempting to schedule transactions of the same type (i.e with the same name on the trace records) in one region for the major transactions. Transactions of the same type often go after similar sets of records. Assigning these to the same region would eliminate a major source of local contention. Transactions of the same type having large total execution times will have to run in more than one region. A table is maintained in the scheduling routine to track the end of the current transaction in each region at every system. The scheduling routine schedules the next transaction and resets the current transaction end time in the region the transaction is to execute, by adding the current transaction execution time to the preceding transaction end time for this region. The scheduling program also adjusts the lock/unlock time of each trace tape record based on the new start time of each transaction. The resulting output is sorted on lock/unlock time and used to drive the trace driven simulation program which determines the resulting lock contention.

To study the buffer invalidation rate and I/O rate, a database block reference trace is needed. A database reference trace can be derived from the lock trace as data entities must be locked before being referenced. Database block references are generated at the time of lock or unlock action for the entity being locked or unlocked. Database blocks are assumed to be modified by a transaction if they are held locked until the end (commit time) of the transaction. Updated blocks are assumed to be written to the disk at the end of the transaction. This is done in IMS from data base recovery considerations.

Trace driven simulation programs are used to simulate the multi-system data sharing environment using the multi-system lock trace and reference trace described above to study the inter-system interference. The simulation program produces a report showing the resources contended for, the contending transactions, the number of contentions for each transaction, buffer invalidation rate, etc.

5. Reduction of Inter-system Interference

In this section, we examine the inter-system interference reduction due to transaction routing based on the trace driven simulations described in Section 4. The effect on global lock contention is examined in Section 5.1, while the effect on buffer invalidation and I/O rate is examined in Section 5.2. Substantial reduction in inter-system interference is observed. Improvement in hierarchical locking to take advantage of affinity based routing is also investigated.

5.1. Global Lock Contention

The effect of routing by affinity on global lock contention is examined. As described in Section 2, block locks are held mainly to prevent simultaneous updates into different records in the same block from transactions executing on different systems. Routing by affinity should substantially reduce the likelihood of updating into the same block from different systems, and hence block lock contentions. For other types of locks, record lock and data set lock, routing transaction accessing the same data into the same system instead of different systems will not eliminate lock contentions. It will only change the contentions from inter-system global lock contentions into intra-system local lock contentions and hence reduce the incurred pass-the-buck protocol overhead, as discussed below. Since the costly inter-system global contentions have been reduced by transaction routing it may become feasible to use optimistic concurrency control methods for inter-system concurrency control and locking for local concurrency control, as we shall see in the next section. Table 5.1 shows the comparison of lock contentions under the two cases, no routing and routing by affinity, for both workloads 1 and 2 on two systems. The columns

Contentions per Transactions

Workload 1

	w/o Routing Block	w/o Routing Other	with Routing Block	with Routing Other
Local	.0006	.0158	.0010	.0523
Global	.0349	.0577	.0024	.0055
Total	.0355	.0737	.0035	.0578

Workload 2

	w/o Routing Block	w/o Routing Other	with Routing Block	with Routing Other
Local	.0025	.0184	.0037	.0568
Global	.0390	.0429	.0110	.0058
Total	.0415	.0612	.0148	.0626

Table 5.1 Contentions for Two Systems

Global Hash Table State Probability

Workload 1

	w/o routing	with routing
0 0	41.3%	40.9%
1 0	56.7%	59.2%
0 1	0.8%	0.19%
1 1	1.2%	0.35%

Workload 2

	w/o routing	with routing
0 0	40.7%	41.5%
1 0	56.1%	57.5%
0 1	0.98%	0.28%
1 1	2.2%	0.67%

Table 5.2 Global Hash Table State Probability for Two Systems

labelled "Block" refer to contentions on block locks, and the columns labelled "Other" refer to contention on record and dataset locks. For example, a contention figure of 0.07 implies that 7% of the transactions will experience blocking during their execution.

We observe substantial reduction in global block lock contentions in both workloads, especially for workload 1. With respect to record and dataset locks, the contentions switch from predominantly global to predominantly local. Total global lock contentions drops to less than one tenth and one fourth of the contentions with no routing for workloads 1 and 2, respectively. The resultant reduction in transaction response time is analyzed in Section 6.

The advantage of the transaction affinity obtained by routing can go beyond the reduction of lock contentions. In a distributed locking environment, a saving in inter-system communication overhead can be obtained as well. One way to reduce the inter-system communications in obtaining global locks is to introduce a control hierarchy on locking. Hierarchical locking discussed below is a slight variation of the scheme described in [GRAY79]. The idea is to map data items into a hierarchy of group items and then try to obtain an exclusive lock on an item at the highest level of the hierarchy containing the target item to be accessed. If unsuccessful, the lock is obtained in shared mode and communication is required with the other system to lock the item at the next highest level of the hierarchy containing the target item. Holding an exclusive lock on an item higher in the hierarchy gives the holder the authority to grant locks to requests on items lower in the hierarchy without further communications to the other systems. The success of hierarchical locking depends upon the existence of locality in lock requests. Intuitively, hierarchical locking would work well in the environment where transactions with affinity are scheduled to execute in the same system. A system can first obtain locks at some high level in the hierarchy, and continue to grant locks at lower level in the hierarchy to different transactions without further inter-system communications. If transactions are routed randomly, the chances of holding locks at high levels in the hierarchy become small and the effectives of hierarchical locking diminishes.

The hierarchical locking structure in IMS is used as a case study to examine the relationship between hierarchical locking and routing by affinity. The findings should be general enough to understand the synergism between the two concepts. IMS uses two hierarchies. The top level of the hierarchy uses locks on hash classes and the next hierarchy uses the record, block or data set locks. As described in Section 2, IMS uses a global hash table (GHT) to track the state of each hash class. If the hash class state is [0,1] or [1,1] the lock request must be communicated to the other system(s) to determine whether the lock may be granted. This would introduce a worst case delay of almost two full pass-the-buck cycles. Affinity based transaction routing reduces substantially the probability that a lock request will encounter either [0,1] or [1,1] state. For workloads 1 and 2 for two systems, Table 5.2 shows that the reduction in the number of lock requests hashing to [0,1] or [1,1] state is by a factor greater than 3. Affinity based routing reduces the tendency of different systems going for the same hash class. However, since [0,1] and [1,1] state probabilities are fairly small even in the case with no routing, the overall improvement in reducing the average wait time for the buck is small. Thus the main effect of affinity based routing is reduced contention.

Somewhat surprising is that the probability of encountering [1,0] state, the state where locks can be granted without inter-system communications, shows little improvement with routing. After careful examination on the cause of [1,0] case, we found that [1,0] hits mainly come from locality within a single transaction not between transactions. The probability that different transactions are going after data in the same hash class simultaneously is quite small. This is consistent with the small value of [0,1] and [1,1] state probabilities observed in the no routing case. Hence, routing is not too helpful in increasing [1,0] state probability. There is still a substantial chance of encountering [0,0] which requires inter-system communication. Although the delay involved is about half to a third of that for [0,1] or [1,1], the frequency of encountering [0,0] is an order of magnitude higher. Hence, the performance is more affected by occurrences of the [0,0] state. One solution to reduce the encountering of [0,0] is to use the notion of hash class retentiveness [SHOE84]. Hash class retentiveness is the tendency that a hash class will be reclaimed by a system after the system releases the ownership on that hash class, i.e. a hash class will be changed directly back to [1,0] after it has been changed from [1,0] to [0,0]. When strong hash class retentiveness exists, it would make sense to let a system continue to keep a hash class in [1,0] state even after it releases all locks in the hash class. The system does not release its ownership on the hash class until a lock is requested in that hash class by some other system. At that time the hash class will be changed into the [0,1] state. This strategy of retaining [1,0] state can significantly reduce the frequency of encountering the [0,0] state. The trade-off is as follows. The system retaining the hash class will not require the communication delay or overhead to change the hash class state when it needs a lock in the hash class the next time. However the delay or overhead for some other system requesting a lock in that hash class is increased since it now observes state [0,1] rather than [0,0]. Let A1 be the total delay or overhead when a [0,0] hash class is encountered and A2 be the total delay or overhead when a [0,1] hash class is encountered. Define the retentiveness ratio to be the ratio of successful retentions to unsuccessful retentions, where a successful retention means that the next request to the hash class is from the system retaining the hash class. A successful retention leads to a [1,0] hit and a saving of A1 whereas an unsuccessful retention leads to a [0,1] hit, and a cost penalty of A2-A1. The hash class retention strategy pays off when the retentiveness ratio is larger than the ratio of A2-A1 to A1. The exact ratio between A2-A1 and A1 depends upon various system parameters, like the buck hold time and communication time. Assuming that the acknowledgement of buck receipt is done from one system to another at the hardware level in negligible time (compared to the buck hold time) it can be shown that the ratio of A2-A1 to A1 should lie between 1 and 0.5, for two systems.

Hash Class Retentiveness

Workload 1		w/o routing	with routing
10 via 00	successful retention	.75	.80
10 via 00	unsuccessful retention	.25	.20
10 via 11	successful retention	.54	.43
10 via 11	unsuccessful retention	.46	.57
Workload 2		**w/o routing**	**with routing**
10 via 00	successful retention	.75	.87
10 via 00	unsuccessful retention	.25	.13
10 via 11	successful retention	.53	.50
10 via 11	unsuccessful retention	.47	.50

Table 5.3 Hash Class Retentiveness For Two Systems

The effect on hash class retention is shown in Table 5.3 for both workloads 1 and 2. The [1,0] state can be reached from either the [0,0] or [1,1] state. The success of the retention, as will be shown later, depends upon how the [1,0] state is reached. In Table 5.3, the row "10 via 00 successful retention" represents the probability that the next request to the hash class will be from the last system owning the hash class after the hash class cycles through [0,0] to [1,0] and back to [0,0], and the row "10 via 00 unsuccessful retention" represents the next request will be from the other system. The row "10 via 11 successful retention" represents the probability that after the hash class reaches [1,0] state through [1,1] and returns to [0,0], the next request will be from the last system owning the hash class, where the row "10 via 11 unsuccessful retention" represents the opposite. Table 5.3 shows that there is a reasonable retentiveness even with no routing for a hash class which reaches [1,0] from [0,0] to be reclaimed by the same system after it has been released. The retentiveness ratio is 3 for both workloads. The retentiveness may be due to IMS trying to schedule transactions which are already loaded. This hash class retentiveness can be further enhanced by transaction routing. With transaction routing by affinity, a hash class cycling from [0,0] to [1,0] and back to [0,0] is reclaimed 80% of the time for workload 1 and 87% of the time for workload 2. This translates into retentiveness ratios of 4 and 6.7 for workloads 1 and 2 , respectively. Also shown in Table 5.3 is the retentiveness of [1,0] when cycling through [1,1] instead of [0,0]. The retentiveness of hash class cycling through [1,1] generally speaking is poor with or without routing. This is because a hash class entering [1,1] state must contain data of interest to both systems. The retentiveness ratio is around 1 in all cases, not enough to justify the retention. Hence in [1,0] retention strategy, [1,0] should be retained only when it does not cycle through [1,1] state. Table 5.4 shows the global hash table state probabilities when [1,0] retention is employed. The [0,0] state probability is substantially reduced, and the [1,0] state probability becomes dominant. Notice that the negative effect of [1,0] retention appears in the increase in [0,1] state probability.

Similar experiments were performed for the three system case for both workloads 1 and 2. The total block lock contentions are again substantially reduced to less than one third of the total block lock

Global Hash Table State Probability

Workload 1	w/o routing	with routing
0 0	8.8%	8.6%
1 0	81.3%	85.4%
0 1	8.6%	5.3%
1 1	1.1%	0.5%
Workload 2	**w/o routing**	**with routing**
0 0	6.0%	6.7%
1 0	81.8%	87.8%
0 1	10.0%	4.8%
1 1	2.2%	0.7%

Table 5.4 Effect of Hash Class Retention on Two Systems

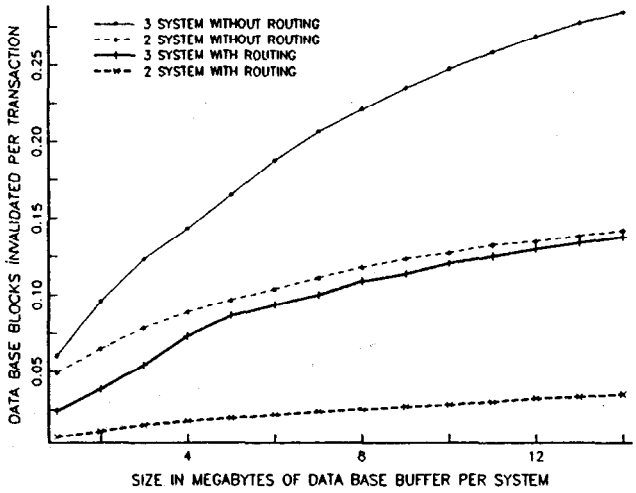

Figure 5.1. Reduction in DB Buffer Invalidation due to Routing.

contentions with no routing. For global locks, the total contentions are again reduced to less than almost one fourth of the contentions with no routing. The retentiveness ratio of [1,0] through [0,0] under affinity based routing remains above 3 and hence hash class retention can still be used to enhance performance, while with no routing the retentiveness ratios may drop below the point where hash class retention is viable.

5.2. Buffer Invalidation and Buffer Hit Ratio

Buffering of data in main memory is used to decrease the number of waits due to I/O. In a multi-system environment, buffer contents can become obsolete due to updates from other systems. IMS uses a buffer invalidation mechanism to insure buffer integrity. A reference to an invalidated block causes an extra read, increasing the I/O rate for multi-systems. Transaction routing based on affinity tends to route transactions making reference to the blocks from the same databases to the same system. The chances of a block in a system's buffer being updated by another system must be smaller as compared to coupling without routing. There is a downward effect on the number of I/O's per transaction due to routing because of a) decreased invalidations and b) because of increased hit ratios - stemming from the fact that transactions making references to the same database tend to be assigned to the same system. Our simulation model captures both effects.

We considered a buffer per system managed by simple LRU replacement policy, for both two and three system coupling with and without transaction routing for workload 1. Invalidated blocks are read from disk if referenced after invalidation.

In Figure 5.1, we plot the number of database buffer blocks invalidated on a per transaction basis as a function of buffer size. The increase with buffer size is due to the fact that larger buffers hold more blocks and each is a candidate for invalidation. In three systems coupling we have more blocks invalidated because for every system there are two other systems that are potential sources of invalidation and in the two system coupling case there is only one other system that may cause buffer invalidation. For a buffer size of 2 Megabytes per system, we note that for three systems coupling the number of blocks invalidated reduces from about 0.09 per transaction to about 0.03 per transaction, and from 0.06 to about 0.01 blocks per transaction for two system coupling.

Since an invalidated block is read from the disk only if rereferenced, the increase in I/O rate is smaller than the invalidation rate. In Figure 5.2 we plot the actual number of reads to the database as a function of the buffer size per system for both two and three coupled systems with and without affinity based routing. As expected, coupling

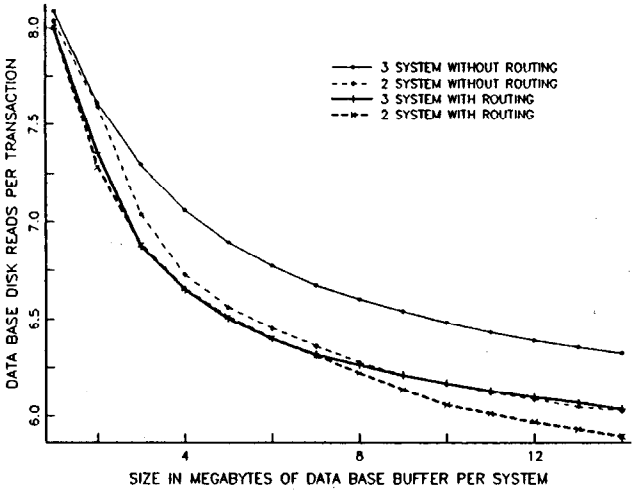

Figure 5.2. Reduction in I/O due to Transaction Routing.

with routing causes less database reads than coupling without routing. For example, for a buffer size of 2M a savings of about 0.25 reads per transaction results. The results from workload 2 were similar and are not presented here.

6. Multi-system Model and Performance

The multi-system model uses as input parameters the lock contention levels, hash-class probabilities, and the reduction in I/O due to the affinity routing, obtained from the trace driven simulations. These parameters are used to project the overall transaction response time with and without affinity routing and hash class retention. The model takes into account the increase in lock contention with transaction rate and buck delay, and the feedback effect of lock contention increasing with response time. The analysis method is first outlined, and is followed by performance projections.

6.1 Approximation for Average Response Time

In [YU85A] an approximate analysis is given to estimate the average response time in the data sharing environment. Related work has been reported in [TAY84], where a mean value analysis methodology, based on the probabilistic access of a database made up of a finite number of granules, has been used. The analysis is validated to within 5% of the simulation results for two systems coupling over a wide range of lock contention probabilities. The transaction response time is expressed as,

$$R \simeq \frac{(R_{CPU} + R_{IO} + R_{BUCK})}{1 - (\frac{P_{CONT}\ L}{3})} \qquad (6.1)$$

where R is the average transaction response time, R_{CPU} is the average time transactions spend at CPU, R_{IO} is the total I/O delay, R_{BUCK} is the time spent in waiting for the buck, P_{CONT} is the lock contention probability, and L is the number of locks per transaction. The reciprocal of the denominator in Equation (6.1) indicates the expansion in the response time due to lock contention wait and is referred to as the lock contention expansion factor. Note that the expansion factor is small for the lock contention levels observed in the trace but will increase substantially at higher contention levels as obtained by increasing the transaction rate. The time R_{IO} is estimated from the average number of I/Os per transaction and the assumed average I/O time, corrected by the reduction in I/O due to transaction routing as estimated in Section 5. The time the transaction spends at the CPU is approximated as follows. Since the average number of locks, unlocks, I/Os, and the frequency of passing the buck, and their concomitant overheads are known, the CPU utilization at each system can be computed, for a fixed lock contention probability. Since the mainframes traced used dyadic CPU's an M/M/2 approximation is used for modelling the (dyadic) CPU and gives,

$$R_{CPU} \simeq \frac{2\ \rho}{\lambda\ (1\ -\ \rho^2)}\ , \qquad (6.2)$$

where λ is the transaction arrival rate per system; R_{CPU} is the time transactions spend at the CPU; and ρ is the CPU utilization per system.

The lock contention probability is proportional to the multiprogramming level, which is the product of the response time and transaction rate. In turn, the response time increases with increasing lock contention. This feedback effect is estimated using an iteration. The transaction response time is first estimated for the contention probability observed in the trace driven simulation (Table 5.1). Then the resulting response time is used to compute a new contention probability by assuming that the contention probability grows as the product of the transaction rate and response time. The approximate model is then run again with the new contention probability to estimate a new response time. The iteration is repeated until convergence is obtained. Only a few iterations are required for the contention range considered.

In optimistic concurrency control schemes [KUNG81], transaction processing proceeds by assuming that no database contention occurs, and if contention is detected later, the transaction is backed out and re-started. The reduction in inter-system contention through affinity based routing suggests that the performance of optimistic type concurrency control will be enhanced. Note that local (within a system) locking is less expensive than global locking (which involves going through the pass-the-buck protocol). Hence, we consider a semi-optimistic concurrency control scheme, where the concurrency control among transactions running on the same system is done through locking and the concurrency control among transactions across systems is done "optimistically". We assume that any transaction requesting a global lock is immediately granted the lock on the assumption that the other systems do not hold that lock concurrently. At the end of the transaction it is determined, by going through the pass-the-buck protocol once, whether the global locks so granted were in conflict with any other on-going transaction on any other system. If a conflict is detected on any global lock the transaction is aborted and after a back-off period (which could be zero) the transaction is restarted from the beginning.

We make the following modifications to our approximate model summarized previously. The probability that a transaction has to restart due to conflict is the transaction conflict probability for global locks that was derived from the trace driven simulations of the previous section. Let us denote this as $L\ P_G$. We assume that a transaction in conflict gives up all its locks and has to acquire them again. We will treat restarted transactions exactly like new transactions. The effective transaction rate input to the model is $\lambda\ /\ 1 - L\ P_G$, where L is the average number of locks per transaction. We have not modelled the following two aspects of restarted transactions a) they may have a higher global lock conflict probability and b) they may have a higher hit probability in the data buffer. The two aspects have opposing effects on the transaction response time. Back-off before restarting a transaction reduces the probability of conflict to that of a new transaction - at the same time the buffers get aged out from the memory during the back-off interval and hence the buffer hit probability will also reduce to that of a new transaction. Hence we treat re-started transactions just like a new transaction. The remaining contention $L\ (P_{CONT}\ -\ P_G)$ is due to local lock contention and denoted as $L\ P_L$. Local contentions are retained in our model and contribute to the expansion in the transaction response time by the denominator in Equation (6.1) where we will use P_L instead of P_{CONT} for the expansion to obtain a nominal transaction response time

$$R_N \simeq \frac{R_{CPU} + R_{IO} + R_{BUCK}}{1 - (\frac{P_L L}{3})} \qquad (6.3)$$

Transaction Rate	20 Trans./Sec/System
Transaction Pathlength	430K instructions
Locks/Transaction	15
Unlocks/Transaction	9
Unlocks at Purge	6
Initialization I/O	5
I/O during Transaction	11
DASD I/O time	35 Milli-Sec.
Buck Delay	5, 20 Milli-Sec.

Table 6.1. Model Parameters

The transaction response time is the nominal response time multiplied by the number of times a transaction has to restart on the average plus the backoff times.

$$R = \frac{R_N}{1 - P_G} + BACKOFF(\frac{1}{1 - P_G} - 1)$$

6.2 Performance of Affinity Routing and Hash Class Retention

We now examine the performance of multi-system data sharing using affinity routing and/or hash class retention, as projected by the model using parametric values found by the IMS trace analysis described in the previous section. Parameter values used are summarized in Table 6.1.

The primary advantage of affinity routing is the reduction in lock contention. This is indicated in Table 5.1, which shows a large decrease in contention with affinity routing, while table 5.2 shows that affinity routing does not affect the hash class probabilities significantly. Since lock contention increases with transaction rate for a fixed response time (and the same database size) affinity routing should be advantageous when this occurs. Figure 6.1 shows response time for two coupled systems when varying the MIPS per system. Here, the transaction rate per MIPS is kept constant to produce a CPU utilization of about 80% , and the buck delay is varied so that the CPU utilization due to buck processing is relatively constant, with a nominal buck delay of 5 msec at 14 MIPS per system. It is assumed that the bandwidth of the I/O system is increased with transaction rate to maintain the average I/O time invariant to system throughput for this mainframe based

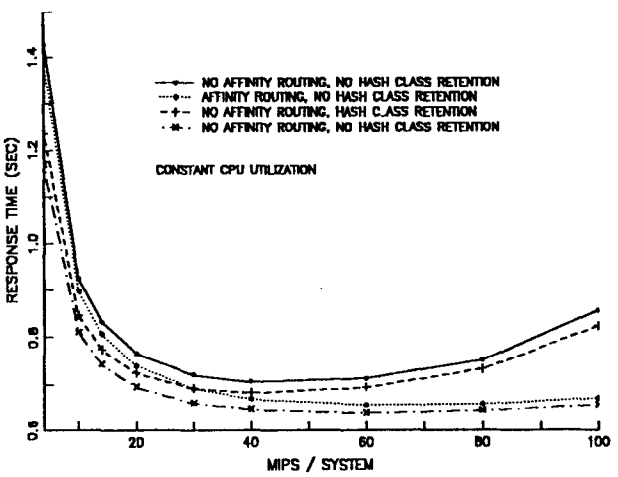

Figure 6.1. Performance: Affinity Routing and Hash Class Retention.

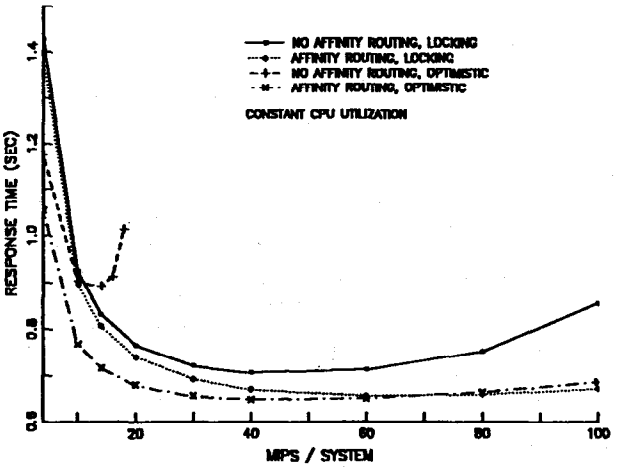

Figure 6.2. Optimistic Concurrency Control with Affinity Routing.

hierarchical database system. This can be achieved by improved buffering in the I/O control unit or by spreading the data over more disks. At low MIPS per system, the contention level is low, and affinity routing produces a small decrease in response time. Hash class retention considerably decreases the probability of the [0,0] hash class, and this results in a significant reduction in response time for low MIPS per system. As the MIPS per systems increases, the transaction rate and lock contention level increase, while the buck delay becomes smaller (since the buck can be passed more frequently for the same absolute CPU overhead). Initially, the response time decreases with increasing MIPS per system because the processing time and buck delay decreases. However, without affinity routing the lock contention increases to a point when it dominates the decrease in processing time and buck delay. Since hash class retention affects the hash class probabilities but does not decrease contention, its benefit becomes smaller. With affinity routing, the lower contention level allows a much larger transaction rate without this lock wait effect becoming predominant.

The effect of affinity routing on the semi-optimistic concurrency control scheme described in Section 6.1 is illustrated in Figure 6.2. The figure also shows how this compares with conventional locking. The assumptions made are identical to those in the previous paragraph. The figure indicates that at low MIPS per system, optimistic concurrency control results in significantly lower transaction response time. This is because the buck delay is larger (for a constant CPU utilization due to buck passing) at low MIPS per system; since with optimistic concurrency control there is no waiting for the buck, the transaction response time decreases. Further, the lock contention level is low in this range, and consequently the transaction restart rate with optimistic concurrency control is small. As the MIPS per system increases, the buck delay can be made smaller and its effect on response time diminishes, while the global lock contention increases leading to a higher transaction restart rate with optimistic concurrency control. Thus, without affinity routing optimistic concurrency control has a worse response time than conventional locking for higher MIPS per system. However, affinity routing reduces the response time for optimistic concurrency control considerably. This is because, affinity routing reduces global lock contention significantly as seen in Table 5.1. Since it is the global lock contention that leads to transaction restarts in this semi-optimistic concurrency control, affinity routing is very effective. Figure 6.2 shows that with affinity routing, semi-optimistic concurrency control does better than conventional locking except at very high MIPS per system. Even in that range, it is the indirect effect of increase in the CPU utilization due to restarts that increases the overall transaction response time.

7. Conclusion

In this paper we developed a methodology to partition transactions into affinity groups and studied the effect of affinity based transaction routing on multi-system data sharing, based on traces from large customers of IBM's IMS database management system. The partitioning technique involves the solution of an integer linear programming problem to determine the affinity groupings. Based on the transaction routing obtained by this technique, a trace driven simulation was run to determine the reduction in lock contention levels, and changes in hash class probabilities for a hierarchical locking scheme. The effect of affinity based routing on buffer invalidation and on the I/O rate is also determined. Finally, an approximate analytical model was used to estimate the overall system performance.

The results of the study indicate that affinity based routing can reduce lock contention considerably. This leads to significant reduction in transaction response time, as the contention increases with total system transaction rate. Affinity based routing also results in reductions in the buffer invalidation rate, and reduction in the I/O. The effect that hash class retention combined with affinity based routing has on reducing the time transactions spend waiting for the buck is then examined. In addition, affinity based routing enbances the performance of the optimistic type of concurrency control considerably.

Acknowledgement

We would like to thank John Robinson for providing clarifications of optimistic concurrency control.

References

[DATE86] Date, C.J., "An Introduction to Database Systems", Vol. 1 and 2, Addison Wesley (1986).

[ESWA76] Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L., "The Notions of Consistency and Predicate Locks in Database Systems", *Comm. ACM 19*, 11 (Nov. 1976), 624-633.

[GRAY79] Gray, J. "Notes on Data Base Operating Systems", in ed. Bayer, R., Graham, R. M. and Seegmuller, G. Operating Systems: An Advanced Course, *Springer-Verlag, New York, 1979*, 394-481.

[KUNG81] Kung, H.T., and Robinson, J.T., "On Optimistic Methods for Concurrency Control", *ACM Transactions on Database Systems, 6*, 2 (June 1981), 213-226.

[SEKI84] Sekino, A., Moritani, K., Masai, T., Tasaki, T., Goto, K., "The DCS - A New Approach to Multisystem Data-Sharing", *Proc. National Computer Conference 1984*, Las Vegas, NV (July 1984).

[SHOE84] Shoens, K., Narang, I., Obermarck, R., Palmer, J., Silen, S., Traiger, I., and Treiber, K., "Amoeba Project", IBM Research Report, RJ4465, San Jose, CA (Oct. 1984).

[STRI82] Strickland, J. P., Uhrowczik, P. P., and Watts, V. L., "IMS/VS: An Evolving System", *IBM Systems Journal 21*, 4 (1982), 490-510.

[TAY84] Tay, Y.C., "A Mean Value Performance Model for Locking in Databases", Ph.D. Dissertation, Harvard University, Cambridge, MA (Feb. 1984).

[WATE67] Waters. L. G., "Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems", *Operations Research, 15, 1171-1174, (1967).*,

[YU85A] Yu, P.S., Dias, D.M., Robinson, J.T., Iyer, B.R. and Cornell, D., "Modelling of Centralized Concurrency Control in a Multi-system Environment", *Performance Evaluation Review 13*, 2 (Proc. 1985 ACM SIGMETRICS Conference), 183-191.

[YU85B] Yu, P.S., Dias, D.M., Robinson, J.T., Iyer, B.R. and Cornell, D., "Distributed Concurrency Control Analysis for Data Sharing", *Proc. 16th Computer Measurement Group Conference*, Dallas, TX (Dec. 1985), 13-20.