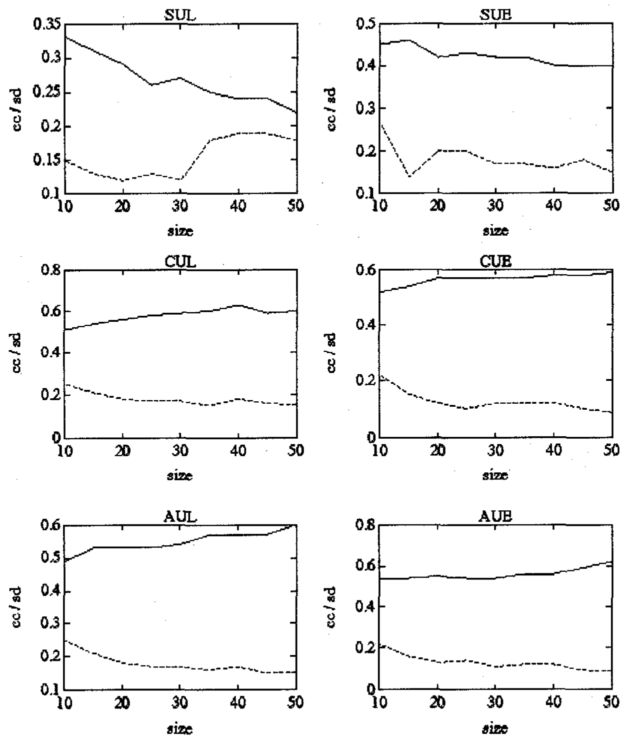

Fig. 3. Ninth type of coefficient of correlation (input size equal to 50).

[7] M.M. Tsangaris and J.F. Naughton, "A Stochastic Approach for Clustering in Object Bases," *Proc. ACM SIGMOD*, pp. 12-21, Denver, May 1991.
[8] A. Bouguettaya, "A Comparative Study of Some Clustering Methods with On-Line Data," M.S. thesis, Dept. of Computer Science, Univ. of Colorado at Boulder, July 1987.
[9] H.C. Romesburg, *Cluster Analysis for Researchers*. London: Lifetime Learning Publications, 1984.
[10] D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Reading, Mass.: Addison-Wesley, 1971.
[11] H.P. Vellandi, "Assymetric Numerically Stratified Cluster Methods," PhD thesis, Dept. of Computer Science, Univ. of Colorado at Boulder, July 1990.
[12] H.C. Boyce, "Mapping Diversity: A Comparative Study of Some Numerical Methods," *Numerical Taxonomy, Proc. Colloquim Numerical Taxonomy*, London, 1969.
[13] P. Sneath, "Evaluation of Clustering Methods," *Numerical Taxonomy, Proc. Colloquim Numerical Taxonomy*, London, 1969.

On Coupling Multiple Systems With A Global Buffer

Ming-Syan Chen, Philip S. Yu, and Tao-Heng Yang

Abstract—In this paper, we conduct a performance study of coupling multiple systems with a global buffer, and present several results obtained from a multiple-system simulator. This simulator has been run against three workloads, and the coupled system behavior with these three different inputs is studied. Several statistics, including those on local and global buffer hits, page writes to the global buffer, cross-invalidations, and castouts are reported. Their relationship to the degree of data skew is explored. Moreover, in addition to the update-caching approach, a design alternative for the use of a global buffer, namely read-caching, is explored. In read-caching, not only updated pages but also pages read by each node are kept in the global buffer, thereby facilitating other nodes' access to the same pages at the cost of a higher global buffer usage. Also investigated is the case of no-caching, i.e., without using a global buffer. Several simulation results are presented and analyzed.

Index Terms—Data sharing approach, global buffer, temporal locality, read-caching.

1 Introduction

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. Among others, one method to couple multiple systems is the data sharing approach [10], which is also referred to as closely coupled clustering by some researchers [6]. Using the data sharing approach, each computing system (or a node) has a local buffer to keep its recently accessed data. For the reason of coherency control, after a page is updated by some node, all other copies of that page in other nodes' local buffers have to be made obsolete, or cross-invalidated. It has been shown that the use of a global buffer shared by all nodes can facilitate each node's access to most recent data copies and improve the overall system 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 load balancing [10]. 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 [3], global buffer management [1], [9], locking [8], and concurrency and coherency protocols [5], [7].

Consequently, IBM announced S/390 parallel offerings, which exploit the above data sharing concept, for its ES9000 systems [2]. In conjunction with this advance, it has become essential to conduct performance study for such a coupled system and to help customers understand the projected performance of various configurations to facilitate their decision on migrating to a proper configuration. Note, however, that most prior work in this context focused on deriving results, via analytical modeling or simulation, for a certain aspect of a data sharing environment. It is noted that

• *M.-S. Chen and P.S. Yu are with the IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598.*
E-mail: {mschen, psyu}@watson.ibm.com.
• *T.-H. Yang is with the IBM Sanata Teresa Laboratory, 555 Bailey Ave., San Jose, CA 95141.*

Manuscript received Apr. 19, 1993; revised June. 6, 1994.
For information on obtaining reprints of this article, please send e-mail to: transactions@computer.org, and reference IEEECS Log Number K96027.

compared to analytical derivations, there are relatively few results based on real customer traces reported on transaction processing behavior in a coupled system. As will be seen, some phenomena which are particularly important to a coupled system, such as temporal locality to be described later, can only be captured in the customer traces. In view of these and the need for performance assessment to cost-effective coupled systems, it is important to study the performance of such systems by evaluating them on representative workloads.

By running a system simulator that is incorporated with very complete functions to model a data sharing environment, we conduct in this paper a comprehensive performance study of coupling multiple systems with a global buffer, and present several results obtained from this trace-driven simulator. This simulator has been run against three workloads, and the coupled system behavior with these three different inputs is studied. The first trace is a synthetic trace that is produced to emulate the OLTP (On-Line Transaction Processing) activities for an inventory control business. The second is generated according to the standard TPC Benchmark A that exercises an update-intensive database service. The third is a customer trace that was taken on a DB2 system [4] from an aircraft company. These three traces cover a wide spectrum of workloads that are of interest for a data sharing environment. A coupled system using a global buffer and a centralized lock manager studied in this paper is shown in Fig. 1. The lock manager employs a hash table to handle the lock granting process. Several statistics, including those on local and global buffer hits, page writes to the global buffer, cross-invalidations (denoted by X-invalidations henceforth for simplicity) and castouts[1], will be comparatively analyzed, and their relationship to the degree of data skew will be explored. By investigating several system component usages, the coupling efficiency and the source of degradation for a coupled system can be determined.

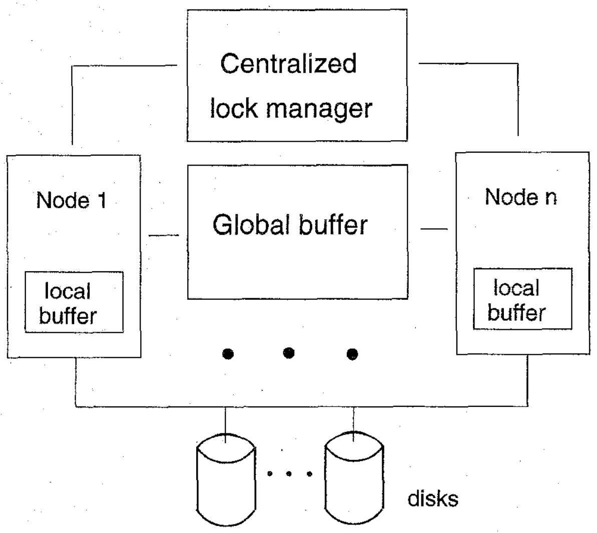

Fig. 1. A coupled system with the data sharing approach.

It is worth mentioning that unlike the situation in synthetic traces and benchmarks, data items (i.e., pages in this study) requested by transactions in the customer trace are not totally independent from one transaction to another. Specifically, consecutive transactions are apt to request the same pages. This can be explained by the fact that one end user is likely to issue several consecutive transactions against the same data record (say, his own account). Such a dependency is termed temporal locality, which can in fact be viewed as one form of data skew for a given time window. Notice that such temporal locality, though not critical to the performance of a single system, has a significant impact on that of a coupled system, since a coupled system with more nodes will benefit less from such locality for its local buffer hits and the corresponding cost for page requests will thus increase. As a result, for better performance of a coupled system, one needs to take into consideration the temporal locality for a good system design.

Moreover, in addition to the standard update-caching approach where a transaction writes its updated pages to the global buffer at the transaction commit time, a design alternative for the use of a global buffer, namely read-caching, is explored. In read-caching, not only updated pages but also pages read by each node are kept in the global buffer, thereby facilitating other nodes' access to the same pages at the cost of a higher global buffer usage. Also investigated is the case of no global buffer, i.e., no-caching. In the no-caching option, locks held by one transaction need to be retained until that transaction commits and all pages updated by that transaction are written to disks. With the above two options available in the data sharing simulator, we analyze their performance tradeoffs in terms of overhead incurred, lock contention, and global buffer hit ratios. It is shown that the read-caching option can improve the global buffer usage significantly, thereby saving a lot of disk I/Os. Also, without using a global buffer, the no-caching option leads to a longer transaction response time and thus to a higher level of lock contention. It is observed that the performance improvement achieved by the read-caching option can be optimized if this option is selectively applied to certain data items based on their access patterns (i.e., read/write ratios).

This paper is organized as follows. A description of the data sharing model considered and the three workloads employed is given in Section 2. Several results, including those on the characteristics of local and global buffers, are presented in Section 3. Two alternatives for the global buffer, read-caching and no-caching, are explored in Section 4. This paper concludes with Section 5.

2 Description of System Model and Workload

Using the data sharing approach, each computing system, or node, has a local buffer to keep its recently accessed data, and all nodes in the system share a global buffer. In response to a page request issued from any of it running transactions, a node tries to find that page in its local buffer (LB) first. If that page is not in its LB, the node will try to get the page from the global buffer (GB). A page which is not in either the LB or the GB will be fetched from disks. For purposes of coherency control, after a node updates a page and writes it to the global buffer at the transaction commit time, all other copies of that page in other nodes' local buffers have to be made obsolete, or X-invalidated. Assuming that a coherency control protocol based on check-on-access is employed [10], when a page is found in the local buffer, one has to check whether that copy is still fresh or not (i.e., not made obsolete by other nodes). If the page is not fresh, it then has to be refreshed either from GB or disks. For a GB write, if there is at least one page made obsolete in any other node's LB, this GB write is said to cause one X-invalidation. Clearly, not every GB write causes a X-invalidation. For illustrative purposes, consider the paradigm in Fig. 2, where there are four nodes and three GB writes from LB1 to pages 38, 17, and 26. It can be seen that the GB write to page 38 will cause one X-invalidation and make one page in LB3 obsolete, whereas the GB write to page 17 makes no page obsolete and thus does not cause any X-invalidation. The dotted box for

1. Castout is the action of writing dirty pages in the global buffer onto disks.

page 17 in the GB means that page 17 was not in the GB prior to this GB write. The GB write to page 26, on the other hand, makes two pages, one in LB3 and the other in LB4, obsolete, accounting for one X-invalidation.

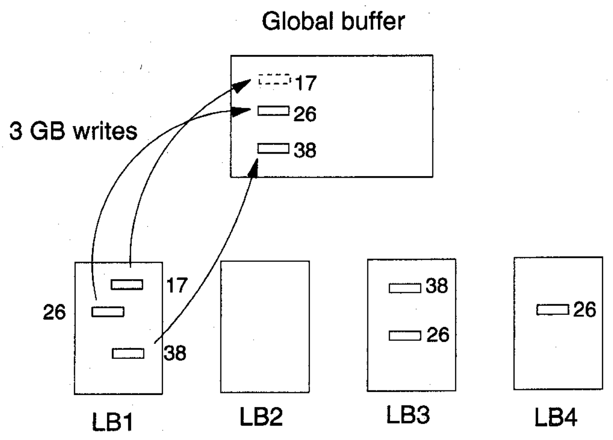

Fig. 2. Illustration of updating three pages from LB1 to the global buffer.

We shall conduct trace-driven simulations against three different workloads, and study the coupled system behavior with these inputs. The system simulator, coded in C++, has been incorporated with very complete functions to model a data sharing environment, particularly for the locking mechanism and LB/GB activities. In our simulation, it is assumed that each node has three processors, each with a processing capability of 18 MIPS. Transactions are assumed to arrive according to a Poisson process with a predetermined arrival rate. The transaction arrival rate is different from one workload to another. Random routing of transactions to nodes is assumed. The locking granularity in our simulation is one page, and the global hash table is assumed to have 9,999 hash entries. Each local buffer, as well as the global buffer, can accommodate 20,000 pages. Pages in the local and global buffers are managed according to the LRU (i.e., Least Recently Used) replacement policy. When the number of dirty pages reaches the corresponding threshold (e.g., 50 percent), a predetermined amount of dirty pages will be cast out from the global buffer to disks. After cast out, a GB page becomes clean again, and stealable for the page replacement policy.

The data sharing simulator has been run against three workloads. The first is a synthetic trace that is produced to emulate the OLTP activities for an inventory control business. The second is generated according to the standard TPC Benchmark A specification. TPC-A is a benchmark that exercises an update-intensive database service. The third is a customer trace which was taken on a DB2 system from an aircraft company during the peak activity period for a duration of 2 hours and 30 minutes. For convenience, these three traces will be referred to as T1, T2, and T3, respectively. As can be seen from its buffer hit ratio and the percentage of X-invalidation per GB write later, T1 has a very skewed data access pattern. Each transaction in average requests 12.5 pages, and updates 1.5 pages. T2 is stated in terms of activities in a hypothetical bank. The bank has one or more branches. One branch has 10 tellers and each teller is associated with 10,000 accounts. The transaction represents the work done by a customer to make a deposit or a withdrawal against an account. T2 is generated according to the TPC-A specification, except that we do not increase the database size according to the number of nodes in the system. Specifically, as the input to our simulator, T2 is generated to have 969 branches, 9,690 tellers and 96.9 million accounts. The database size is fixed throughout all simulations so that the effects relevant to the system capacity can be properly observed and fairly compared to those from T1 and T3. Also, in contrast to the skewed access in T1 and T3, T2 is generated so as to have a uniform data access pattern. Each transaction in T2 requests 11 pages and updates four pages. T3, the customer trace from an aircraft company, also shows a very skewed data access pattern. In average, each transaction in T3 requests 11.3 pages, and updates 0.8 pages.

3 Performance of a Coupled System

In this section, the overall performance of a coupled system with a global buffer is studied. Several statistics on the characteristics of local and global buffers are analyzed comparatively, and their relationship to the degree of data skew is explored.

3.1 Local and Global Buffer Hit

In a single system, the local buffer hit ratio depends mainly upon the local buffer size and the data access pattern. In a multinode environment, however, the local buffer hit ratio is also affected by the X-invalidation phenomenon. The local buffer hit ratios from the three workloads are shown in Fig. 3, where the number of nodes in the coupled system ranges from two to six. Results for a single system are also given for comparison purposes. It can be seen from Fig. 3 that the synthetic trace T1 and the customer trace T3 have approximately the same degree of data skew and show fairly high local buffer hit ratios. The local buffer hit ratio of T2, on the other hand, stays around 60% because of the uniform generation. Note that LB hit ratios decrease as the number of nodes increases, due to the effect of X-invalidation.

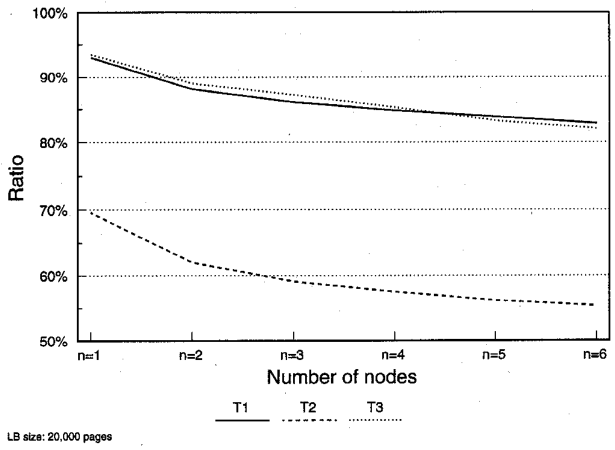

Fig. 3. Local buffer hit ratios for the three workloads.

The distribution of page requests after a buffer miss for T1 is shown in Fig. 4 where each stacked bar consists of three components, pages obtained from disks, pages obtained from GB and pages refreshed from GB. Recall that a page is refreshed from the GB if that page is in the LB but has been made obsolete by others. It is observed that the number of pages refreshed from GB increases with the number of nodes in the system. This can be explained by the fact that the more nodes in the system, the more likely a local buffer page would be made obsolete by transactions running in other nodes, thus increasing the number of pages refreshed from the GB. The distribution of page requests after a buffer miss for T2 is similar to the one for T1, and is thus omitted here. The distribution of page requests after a buffer miss for T3 is shown in Fig. 5, where it can be seen that the portion of pages obtained from the GB is less than that in Fig. 4, due to a relatively smaller write ratio for transactions in T3.

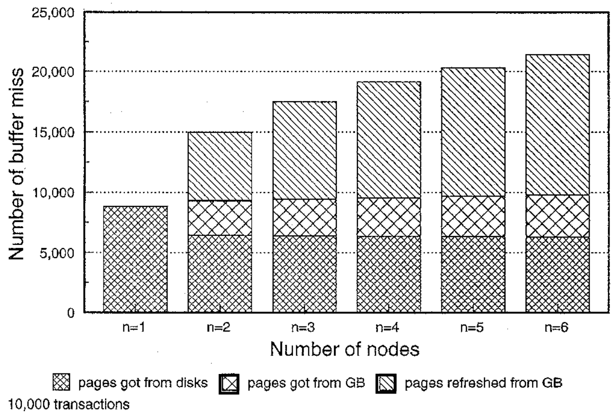

Fig. 4. The local buffer miss for trace T1.

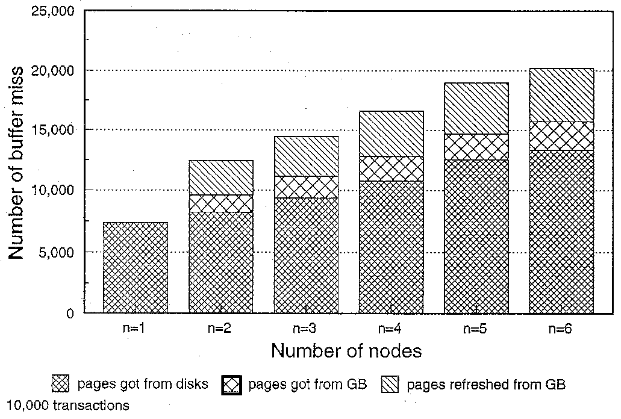

Fig. 5. The local buffer miss for trace T3.

It is important to see that, despite their similarity in the degree of data skew, there is a fundamental difference between the synthetic trace T1 and the customer trace T3. Note that the pages from 1) local buffer hits and 2) refreshed from GB can be thought of as those that would be from local buffer hits if there were no X-invalidation phenomenon. It can be observed from Fig. 4 that the total number of pages obtained from disks and GB remains approximately the same for all numbers of nodes, whereas such a number increases with the number of nodes in Fig. 5. This shows the difference between T1 and T3. Unlike those in the synthetic trace T1, data pages requested by transactions in the customer trace T3 are not totally independent from one transaction to another because consecutive transactions are apt to request the same pages due to temporal locality. Clearly, temporal locality increases the local buffer hit ratio. Notice, however, that since random transaction routing is employed in our trace driven simulation for load balancing, a system with more nodes will benefit less from such temporal locality for its local buffer hits.

3.2 Characteristics of GB Writes: X-Invalidations and Castouts

As mentioned earlier, after a node updates a page to GB, all other copies of that page in other nodes' local buffers have to be made obsolete. Clearly, the percentage of X-invalidations depends on the degree of data skew, and affects the local buffer usage. The more skewed the data are, the larger this percentage will be. A skewed input will increase the LB hit ratio as well as the effect of X-invalidation. Thus, unlike the situation in a single system, the impact of data skew to the LB hit ratio in a data sharing environment is a result from two opposite effects. Furthermore, the average number of cast out pages per GB write is also affected by the skew factor. Note that in a data sharing environment, every GB write does not necessarily update a clean page. A clean page is a page that has not been changed since last cast out. Specifically, some GB writes might update pages that have been changed (i.e., set dirty) since last cast out. As a result, such GB writes do not increase the number of pages to be cast out to disks later. For the example in Fig. 2, suppose page 38 was dirty and page 26 was clean in the GB prior to the three GB writes from LB1. Then, after these three GB writes in Fig. 2, we have two more pages (i.e., pages 17 and 26) for cast out. Since one cast out page may capture changes by many GB writes, the average number of cast out pages per GB write is between zero and one. The more skewed the data, the smaller this number will be. Note that the number of cast out pages per GB write is not only useful to derive the cast out threshold and optimize the tradeoff between system performance and recovery efficiency, but is also important to determine the global buffer size required.

The percentage of X-invalidations and the average number of cast out pages per GB write for T1 are shown in Fig. 6. It can be seen that while the former slightly increases with the number of nodes in the system, the latter stays at about 28 % for all numbers of nodes. This agrees with our intuition since the more pages in the local buffers of other nodes, the more likely a GB write would cause a X-invalidation. On the other hand, the average number of cast out pages per GB write is actually dependent on the cast out threshold, and immaterial to the number of nodes in the system. In the simulation, the cast out threshold is independent of the number of nodes in the system. Fig. 6 indicates that the data access pattern in T1 is very skewed. The average numbers of pages made obsolete per X-invalidation are shown in Fig. 7.

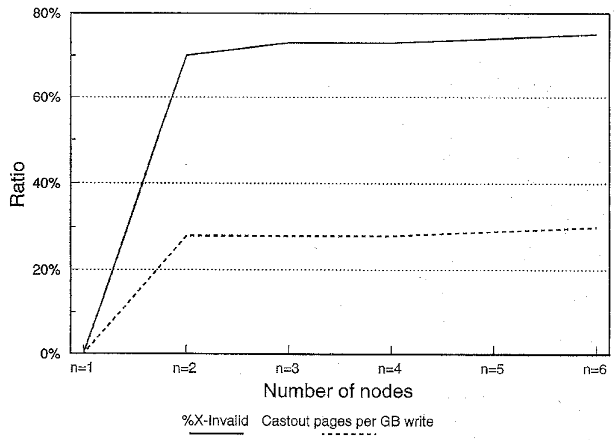

Fig. 6. Characteristics of writes to the global buffer for trace T1.

4 Effect of Read-Caching and No-Caching

Note that the global buffer is employed to store updated pages by transactions thus far. This design is called update-caching. In this section, we shall explore two design alternatives for the use of GB, read-caching and no-caching. In read-caching (RC) option, similarly to updated pages, pages read by each node are also kept in the global buffer. In no-caching (NC) option, the global buffer is used for X-invalidation purposes only, not for storing updated pages. Specifically, while the original design caches updated pages in the GB, RC caches *both* updated *and* read

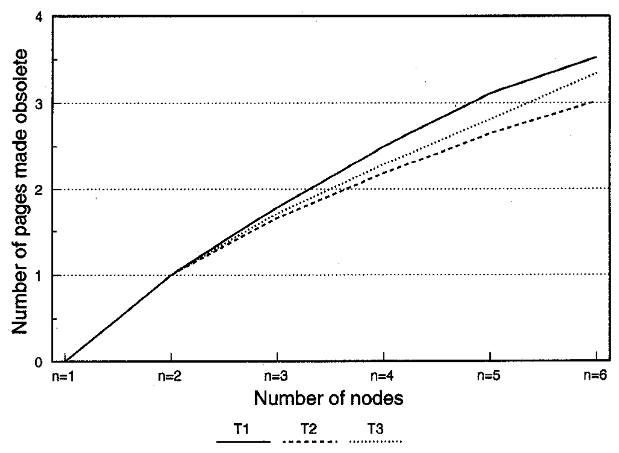

Fig. 7. Average number of pages made obsolete per X-invalidation.

pages in the GB, and NC caches *neither* updated *nor* read pages. Note that the concurrency control protocol operates in accordance with the corresponding operation (i.e., read or write).

4.1 The Option of Read-Caching

Under the RC option, not only updated pages but also pages read by each node are kept in the global buffer. It is observed from traces that some pages, though very frequently accessed, are never updated (e.g., some index pages in T2), and thus not stored in the global buffer. Clearly, keeping such pages in the GB will facilitate other nodes to obtain them and reduce disk I/Os at the cost of a higher GB usage. In addition, as pointed out earlier, the advantage from the existence of temporal locality found in customer traces can be compromised by a transaction routing strategy aiming at good load balancing among all nodes. To exploit temporal locality, one may want to store some read-miss pages in the GB so that pages requested by other nodes, even for read purposes, can also be available in the GB. In view of this, it is important to explore the approach of read-caching in a coupled system to take full advantage of the use of GB and minimize the disk I/Os.

It is noted that while helping the GB hit ratios and saving disk I/Os, the RC option significantly increases the GB usage and also the LRU page replacement activities in the GB. This suggests that a larger GB size be required for adopting the RC option. It is also observed from traces that different data items may have very different access patterns. Specifically, the index pages, which are infrequently updated, tend to be more skewed than data pages, thus being good candidates for the RC option. The effect of read-caching on GB hit for trace T1 is shown in Fig. 8, where three alternatives, no read-caching[2], read-caching for index pages, and read-caching for all pages, are investigated. It can be seen that the RC option can improve the GB hit ratios (including pages with GB hit and those refreshed from the GB) significantly, thereby saving a lot of disk I/Os. Fig. 8 also shows that due to the characteristics of index pages, it is still very beneficial even if the RC option is only applied to index pages. The effect of RC, applied to both index and data pages, on the CPU cost incurred per transaction is shown in Fig. 9. Here the ordinate shows the cost related to RC, which is normalized by the corresponding transaction path lengths. By paying an extra cost of GB read/write for some pages, the RC option saves the corresponding disk I/Os and leads to overall improvement. It is also observed from the results in Fig. 9 that T3 benefits more from the RC option than T1. Note that, as mentioned in Section 4.1, T3 has a smaller write ratio than T1, and thus benefits less from the use of GB in the original update-caching design. Such a disadvantage is offset by the RC option, explaining why T3 could achieve more improvement from the RC option than T1.

2. Note that no read-caching in Fig. 8 means the original design of update-caching, and should not be confused with the no-caching option.

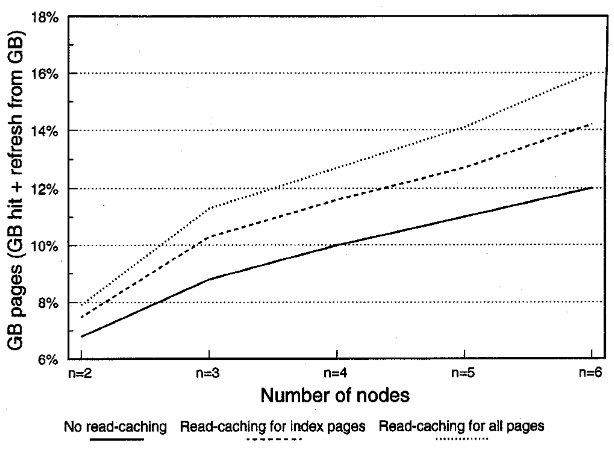

Fig. 8. The effect of read-caching on GB hit for trace T1.

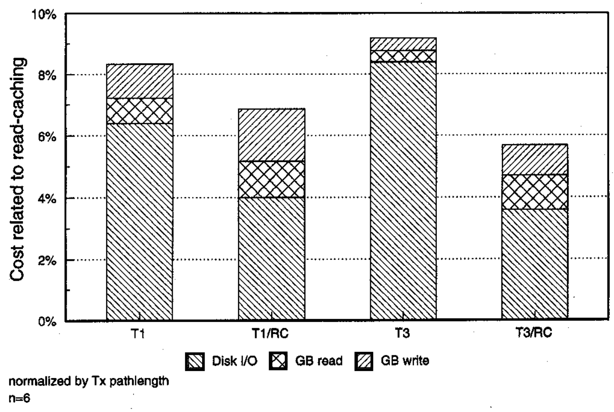

Fig. 9. The effect of RC on the extra CPU cost per transaction.

4.2 The Option of No-Caching

Since updated pages are not kept in the global buffer under this option, for the reason of concurrency control, locks held by one transaction thus need to be retained until that transaction commits and all pages updated by that transaction are written to disks. This unavoidably leads to a much longer transaction response time, and then a higher level of lock contention due to a longer average lock holding time. For a low transaction arrival rate, the effect of NC on lock contention for T3 is studied. It is seen that even for a low transaction arrival rate, the NC option clearly suffers a higher degree of lock contention for all numbers of nodes. When high throughput is needed, the NC option would become even more unfavorable, because the maximal transaction arrival rate that a coupled system could stand will be drastically reduced by the NC option due to its longer transaction response time and higher level of lock contention. The effect of NC on the extra CPU cost incurred per transaction is shown in Fig. 10, where the difference caused by the NC option, as opposed to the original update-caching approach, is shown. Four cases are evaluated on the abscissa, and the ordinate shows the extra CPU cost (in terms of the number of instructions normalized by the corresponding transaction path lengths) caused by the NC option. The numbers in parentheses denote negative numbers (i.e., the CPU cost *saved* by the NC option), and the actual extra cost by the NC option is determined by taking both positive and negative numbers into account.

From the four cases shown in Fig. 10, it can be seen that the NC option incurs a significant amount of CPU cost in disk I/O and some in lock contention, while yielding a moderate savings in GB read/write. It is observed that the amount of extra cost by the NC option increases with the number of nodes in the system for both T1 and T3, meaning that as the number of nodes increases it is more advantageous to have a global buffer. This agrees with our intuition. Recall that the X-invalidation effect increases with the number of nodes. The use of a GB can in fact remedy the X-invalidation phenomenon in that most obsolete pages can be refreshed from the GB, hence saving costly disk I/Os that would be required to fetch those pages from disks. In addition, it is noted that the extra cost of the NC option in T3 is smaller than that in T1. Note that T3 has fewer pages updated, thus fewer GB writes per transaction than T1, implying that T3 benefits less from the existence of a GB than T1 in the original update-caching design. Performance of T3 is thus less affected by the NC option.

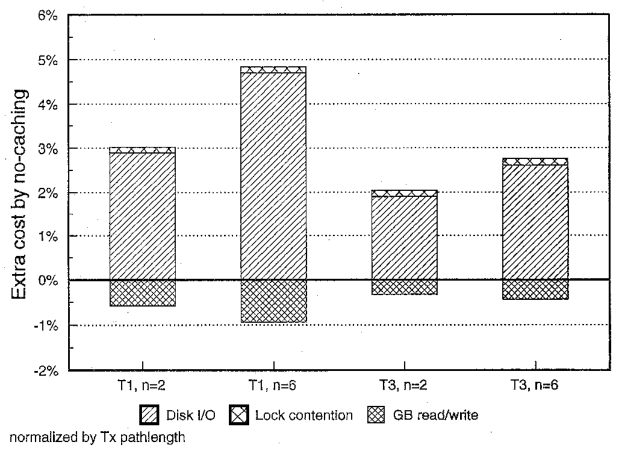

Fig. 10. The effect of NC on the extra CPU cost per transaction.

5 Conclusions

In this paper, a performance study of coupling multiple systems with a global buffer was conducted, and several results from a comprehensive data sharing simulator were presented. This simulator has been run against three workloads, and the coupled system behavior with these three different inputs studied. Moreover, the read-caching option was explored as an alternative to update-caching for the use of global buffer. It is observed that the performance improvement achieved by the read-caching option can be optimized if this option is selectively applied to certain data items. Also, it is shown that without using a global buffer, the no-caching option leads to a longer transaction response time and thus a higher level of lock contention. Several simulation results have been presented and analyzed, and some important factors for the performance of a coupled system, including the temporal locality that exists in some customer traces, have been identified and explored.

Acknowledgments

The authors would like to thank S. Shyu and T. Liu for their contributions in the early stages of this study.

References

[1] M.J. Carey, M.J. Franklin, M. Livny, and E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architectures," *Proc. ACM SIGMOD*, pp. 357-366, May 1991.

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

[3] A. Dan, P.S. Yu, and D.M. Dias, "Performance Modelling and Comparison of Global Shared Buffer Management Polocies in A Cluster Environment," *IEEE Trans Computers*, vol. 43, no. 11, pp. 1,281-1,297, 1994.

[4] D.J. Haderle and R.D. Jackson, "IBM Database 2 Overview," *IBM Systems J.*, vol. 23, no. 2, pp. 112-125, 1984.

[5] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Trans. Database Systems*, vol. 17, no. 1, pp. 94-162, Mar. 1992.

[6] E. Rahm, "Evaluation of Closely Coupled Systems for High Performance Database Processing," *Proc. 13th Int'l Conf. Distributed Computing Systems*, pp. 301-310, May 1993.

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

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

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

[10] 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.