Affinity-based Dynamic Transaction Routing in Shared Disks Clusters

Kyungoh Ohn Haengrae Cho Byoungchul Ahn

Department of Computer Engineering, Yeungnam University
Kyongsan, Kyongbuk 712-749, Korea

Abstract

Coupling multiple nodes for high performance transaction processing has become increasingly attractive for reasons of capacity, availability and cost. The shared disks (SD) cluster couples computing nodes via a high-speed network and share a common database at the disk level. This paper proposes a transaction routing algorithm that exploits access affinities of transactions to rationalize workload on computing nodes in the SD cluster. The proposed algorithm is novel in the sense that it can achieve an optimal balance between affinity-based routing and dynamic load balancing. The simulation results show that the proposed algorithm exhibits substantial performance improvement when transaction workload is changed dynamically.

Keywords: cluster computing, shared disks, transaction routing, performance evaluation

1 Introduction

A cluster is a collection of interconnected computing nodes that collaborate on executing an application and presents itself as one unified computing resource [12]. Clusters have been used in a variety of environments including on-line transaction processing, e-commerce, and parallel database systems such as IBM DB2 Parallel Edition [4] and Oracle9*i* Real Application Cluster [9]. Depending on the nature of disk access, there are two primary flavors of cluster architecture designs: *shared disks* (SD) and *shared nothing* (SN) [12]. The SD cluster allows each node to have direct access to all disks. In the SN cluster, however, each node has its own set of private disks and only the node can directly read and write its disks. The SD cluster offers a number of advantages compared to the SN cluster, such as dynamic load balancing and seamless integration, that make it attractive for high performance transaction processing. Furthermore, the rapidly emerging technology of *storage area networks* (SAN) makes SD clusters the preferred choice for reasons of higher system availability and flexible data access [9].

In the SD cluster, all nodes are connected via a high-speed network and share a common database at the disk level. Each node in the SD cluster has its own buffer pool and caches database pages in the buffer. Caching may substantially reduce the number of disk I/O operations by utilizing the locality of reference. However, since a particular page may be simultaneously cached in different nodes, modification of the page in any buffer invalidates copies of that page in other nodes. This necessitates the use of a *cache coherency scheme* so that the nodes always see the most recent version of database pages [2, 7].

The subject of this paper is about transaction routing that corresponds to selecting a node for an incoming transaction to be executed. In the SD cluster, if transactions referencing similar data are clustered together to be executed on the same node (*affinity node*), then local buffer hit ratio should increase and the level of interference among nodes due to buffer invalidation and lock synchronization will be reduced. This concept is referred to as *affinity-based routing* [8, 13].

Even though the affinity-based routing may reduce transaction response time, it is purely a static one. It is very much non-adaptive to the changes in the system load, that is, it does not take the current load distribution of each node into account while taking the routing decisions. This is particularly problematic when the load deviation of each node is quite large. For example, if transactions referencing specific database partition are congested, then the corresponding affinity node will be overloaded while other nodes are idle. To alleviate this problem of affinity-based routing, a *dynamic load balancing algorithm* should be considered together so that the mapping of a transaction class to the affinity node(s) can be changed dynamically.

In this paper, we propose a new affinity-based dynamic transaction routing algorithm, named *Dynamic Affinity Cluster Allocation* (DACA). DACA is novel in the sense that it can maximize the effect of load balancing while reducing the negative effect of frequent buffer invalidation.

The rest of this paper is organized as follows: Section 2 summarizes the related works. Section 3 presents the details of DACA. We have evaluated the performance of DACA under a number of database workloads. Section 4 describes the experiment model, and Section 5 analyzes the experiment results. Finally, Section 6 presents the conclusions.

2 Related Works

There are few studies on affinity-based dynamic transaction routing in the SD cluster [8]. All of related studies select the execution node of incoming transactions based on data affinity when the system load is evenly balanced. However, there are some differences on handling overload due to the congestion of some transaction class. In [14], the incoming transactions of the congested class are spread across *all* nodes. On the other hand, in [5], the transactions are routed to the *least loaded* node. Both studies may achieve load balancing dynamically. However, the performance improvement could be marginal since the buffer hit ratio of each node decreases. The reduction comes from two factors [14]. First, in each of the nodes except the surge node (i.e., the node corresponded with the congested class) the granules in the buffer comes from two partitions: the original partition logically assigned to the node and the partition corresponding to the congested class. The second is the frequent cross invalidation effect on the granules of the partition corresponding to the congested class.

Recently, Oracle9*i* Real Application Cluster introduces the notion of *dynamic resource remastering* to reduce the locking overhead when affinity-based routing is adopted [9]. The basic idea is to locate the locking information of a database partition to the affinity node where the database partition is most frequently accessed; hence, lock operations on the database partition can be processed locally. However, the dynamic resource remastering is just a mechanism to exploit the advantage of affinity-based routing, but does not imply any transaction routing algorithms.

It is worthy to compare the transaction routing with the task scheduling, which was studied extensively in general distributed systems [1, 6, 11]. These studies usually assumed that each task can be processed in its entirety by any node and it requires CPU and memory resources. Some dynamic load balancing algorithms often migrate tasks to other node during their execution. However, in transaction routing, migrating a transaction between nodes are not make sense, since most transactions execute in short duration [8]. Furthermore, transaction routing is more complex, since system performance is affected by additional factors such as disk I/O, data contention due to concurrent access, and bookkeeping overhead for database recovery [3].

3 Proposed Algorithm

We propose a new affinity-based dynamic transaction routing algorithm, named *Dynamic Affinity Cluster Allocation* (DACA). The goals of DACA are two-fold: (a) to maximize the effect of load balancing and (b) to alleviate the routing overhead. To achieve the first goal, DACA balances the load of each node so that the resulting buffer hit ratio should not decrease significantly. DACA achieves the next goal by adopting the notion of *affinity cluster* (AC) [13] that collects several transaction classes with high affinity to a given set of relations. The routing algorithm can be simplified by considering to balance the load of ACs not each transaction class.

3.1 Preliminary

Table 1 summarizes the routing parameters of DACA. We assume that #AC is equal to or smaller than #N so as to minimize the load differences among ACs [13]. A *transaction router* maintains the routing parameters and routes incoming transactions to nodes. Specifically, when a transaction router allocates a transaction of AC_i to a node N_j, it increments both $\#\mathrm{T}(\mathrm{AC}_i)$ and $\#\mathrm{T}(N_j)$. When N_j informs the commitment of the transaction to the router, both counters are decremented.

The transaction router implements a *routing function*, $\mathcal{R}$, which specifies the set of nodes allocated for each AC. $\mathcal{R}^{-1}$ is an inverse function of $\mathcal{R}$. If $\mathcal{R}(\mathrm{AC}_i)$ includes several nodes, incoming transactions of AC_i are routed to one of the nodes in a round-robin fashion. Initially, $\mathcal{R}(\mathrm{AC}_i)$ is set to $\{N_i\}$, which means that transactions of AC_i is routed to N_i. As a result, for every AC_q and N_p, both $\mathrm{Card}(\mathcal{R}(\mathrm{AC}_q))$ and $\mathrm{Card}(\mathcal{R}^{-1}(N_p))$ are set to 1 initially. It is possible to optimize the initial setting by assigning more nodes to some AC, if we expect that transactions of the AC will occupy large portion of the system load.

DACA divides the type of overload into AC overload and node overload. The AC overload implies the case where transactions of some AC_q are congested. The node overload occurs when a node N_p ia allocated to several ACs and the total number of transactions executed at N_p is over average. In the followings, we define several terminologies to model the status of an AC and a node.

Definition 1 (*AC Overload*) An affinity cluster AC_q is overloaded, if $\mathrm{Card}(\mathcal{R}(\mathrm{AC}_q)) < \#\mathrm{N}$ and $\#\mathrm{T}(\mathrm{AC}_q) \;/\; (\mathrm{Card}(\mathcal{R}(\mathrm{AC}_q)) + 1) \geq \bar{L}(N)$.

Definition 2 (*Node Overload*) A node N_p is overloaded, if $\mathrm{Card}(\mathcal{R}^{-1}(N_p)) > 1$ and $\#\mathrm{T}(N_p) \geq \bar{L}(N) \times \alpha$, where α is a sensitivity factor and is equal to or greater than 1.

Table 1: Routing Parameters

Parameter	Description
#AC	Number of ACs
#N	Number of nodes (#N $\geq$ #AC)
Mem(N_p)	Memory size of a node N_p
HotSet(AC_q)	Size of hot set accessed by an affinity cluster AC_q
#T(N_p)	Number of transactions executing at N_p
#T(AC_q)	Number of executing transactions of AC_q
$\mathcal{R}(AC_q)$	Set of nodes allocated for AC_q
Card($\mathcal{R}(AC_q)$)	Cardinality of $\mathcal{R}(AC_q)$
$\mathcal{R}^{-1}(N_p)$	Set of ACs allocated to N_p
Card($\mathcal{R}^{-1}(N_p)$)	Cardinality of $\mathcal{R}^{-1}(N_p)$
$\bar{L}(N)$	Average load of nodes ($\Sigma_{i=1}^{\#N}$(#T(N_i)) / #N)

Definition 3 (*AC Underload*) An affinity cluster AC_q is underloaded, if Card($\mathcal{R}(AC_q)$) > 1 and #T(AC_q) / (Card($\mathcal{R}(AC_q)$) $- 1$) $< \bar{L}(N)$.

3.2 Transaction Routing

DACA balances the load of each node according to the overload type. If AC_q is overloaded, then DACA allocates one more node to AC_q by expanding $\mathcal{R}(AC_q)$. We refer this strategy as *node expansion*. On the other hand, if there are no AC overloads but N_p is overloaded, then DACA distributes some ACs in $\mathcal{R}^{-1}(N_p)$ to other node. We refer this strategy as *AC distribution*. Finally, if AC_q is underloaded, then DACA excludes some node allocated to AC_q by reducing $\mathcal{R}(AC_q)$. This strategy is referred as *node reduction*. Now we describe each strategy in detail.

3.2.1 Node Expansion

Suppose AC_q is overloaded and $\mathcal{R}(AC_q)$ is $\{N_i\}$. Since routing new incoming transactions of AC_q to N_i suffers from increased response time, the transaction router changes $\mathcal{R}(AC_q)$ to include additional node. We select the *least* loaded node, say N_k, as a candidate. This means that #T(N_k) is minimal among other nodes. Then $\mathcal{R}(AC_q)$ is extended to $\{N_i, N_k\}$, and incoming transactions of AC_q is routed to either N_i or N_k in a round-robin fashion.

A complicated case occurs when N_k has already been assigned to some affinity cluster, say AC_k. In this case, we first check whether AC_k is underloaded. If that is true, a node reduction strategy is applied so that $\mathcal{R}(AC_k)$ excludes N_k, and then N_k is assigned to AC_q. We describe the node reduction strategy in Section 3.2.3. If AC_k is not underloaded, $\mathcal{R}(AC_k)$ is changed to route transactions of AC_k to other node. The next section describes how to select a new node to execute transactions of AC_k. At both cases, AC_q becomes the only affinity cluster allocated to N_k.

An important point to note about this strategy is that it tries to reduce the number of nodes allocated for each affinity cluster as far as the load deviation is not significant. This is not true in previous routing algorithms [5, 14], where transactions of overloaded AC could be routed *immediately* across all nodes. Since transactions of an AC are routed to several nodes in a round-robin fashion, each local buffer should cache similar database pages. As a result, there would be a lot of replication of database pages among the nodes and buffer invalidation may be occurred frequently. Frequent buffer invalidation results in lower local buffer hit ratio and lots of message passing to transfer database pages between nodes. As more nodes execute transactions of an AC, the buffer invalidation effect should limit the potential performance improvement of dynamic load balancing.

3.2.2 AC Distribution

AC distribution strategy is applied when a node N_p is overloaded, and thus $\mathcal{R}^{-1}(N_p)$ includes several ACs. Among the ACs, suppose that #T(AC_s) is the minimum and #T(AC_l) is the maximum. If there is a node N_k not allocated to any AC, i.e. $\mathcal{R}^{-1}(N_k) = \{\}$, then the transaction router updates $\mathcal{R}(AC_l)$ to $\{N_k\}$. Similarly, if N_k is allocated to an underloaded AC, the transaction router also updates $\mathcal{R}(AC_l)$ to $\{N_k\}$ after performing the node reduction procedure.

Otherwise, the transaction router updates $\mathcal{R}(AC_s)$ to include other node, say N_k, where it is not overloaded and $\mathcal{R}^{-1}(N_k)$ is not empty. Suppose that $\mathcal{R}^{-1}(N_k)$ is currently set to $\{AC_i\}$. If we route both AC_i and AC_s to N_k, the local buffer of N_k caches all database pages accessed by them. Since different ACs would access disjoint portion of database, frequent buffer miss could occur if the local buffer is not large enough to cache the hot sets of both ACs. To avoid frequent buffer miss, AC_s is routed to N_k only if HotSet(AC_i) + HotSet(AC_s) $<$ Mem(N_k). If there

are one or more nodes satisfying the constraint, the transaction router allocates AC_s to the least loaded node among them. If there is no node satisfying the constraint, AC_s is allocated to the least loaded node in the system.

3.2.3 Node Reduction

Suppose that AC_q is underloaded and $\mathcal{R}(AC_q)$ is $\{N_i, N_k\}$. Then the transaction router excludes one of the nodes from $\mathcal{R}(AC_q)$. Selecting the excluded node depends on the existence of other ACs assigned to them. If one of the nodes, say N_k, is assigned to other ACs, we exclude N_k from $\mathcal{R}(AC_k)$. Otherwise, it is possible to exclude any node from $\mathcal{R}(AC_q)$. Similar to the node expansion strategy, this strategy of excluding nodes comes to satisfy the first goal of DACA. Transactions of the same AC should be routed to the smaller number of nodes when the resulting routing strategy does not incur the AC overload.

4 Simulation Model

We have compared the performance of DACA with other routing algorithms under a wide variety of database workloads and system configurations using an SD cluster simulation model. Figure 1 shows the simulation model of SD cluster. The simulation model was implemented using the CSIM discrete-event simulation package [10].

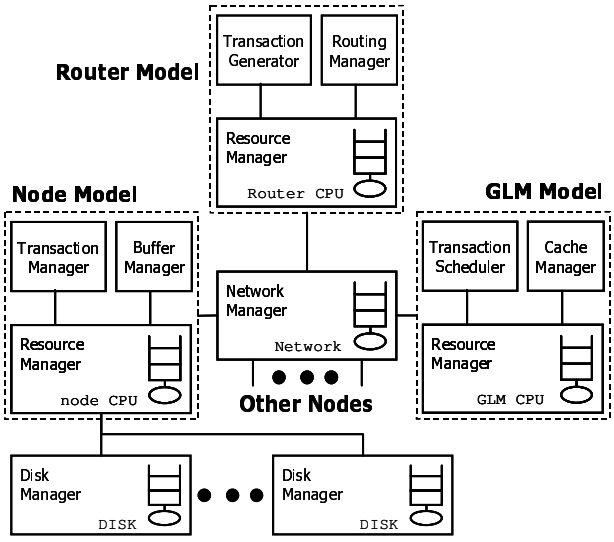

Figure 1: Simulation Model of SD cluster

We model the SD cluster consisting of a single router and a global lock manager (GLM) plus a varying number of nodes, all of which are connected via a local area network. The router model consists of a *transaction generator* and a *routing manager*. The transaction generator has a role to generate transactions, each of which is modeled as a sequence of database operations, i.e., each lock request is followed by a database access operation (a read or a write). The routing manager captures the semantics of a given transaction routing algorithm.

Table 2: Simulation Parameters

System Parameters		
LCPUSpeed	Instruction rate of node CPU	50 MIPS
GCPUSpeed	Instruction rate of GLM CPU	100 MIPS
NetBandwidth	Network bandwidth	100 Mbps
NumNode	Number of computing nodes	1 – 16
NumDisk	Number of shared disks	10
MinDiskTime	Minimum disk access time	0.01 sec
MaxDiskTime	Maximum disk access time	0.03 sec
PageSize	Size of a page	4096 bytes
RecPerPage	Number of records per page	10
ClusterSize	Number of pages in a cluster	10000
HotSize	Size of hot set in a cluster	2000
DBSize	Number of clusters in database	8
BufSize	Per-node buffer size	4000
MsgInst	Number of instr. per message	22000
LockInst	No. of instr. per lock/unlock pair	2000
PerIOInst	Number of instr. per disk I/O	5000
PerObjInst	Number of instr. for a DB call	15000
LogIOTime	I/O time for writing a log record	0.005 sec
Transaction Parameters		
TrxSize	Number of pages accessed by a trx.	10
SizeDev	Transaction size deviation	0.1
WriteOpPct	Probability of updating record	0.2
MPL	Number of concurrent transactions	10 – 640
ACNum	Number of ACs	1, 8
ACLocality	Probability of accessing local cluster	0.8
HotProb	Probability of accessing hot set	0.8

The model for each node consists of a *buffer manager*, which manages the node buffer pool using an LRU policy, and a *resource manager*, which models CPU activity and provides access to the shared disks and the network. For each transaction, the *transaction manager* forwards lock request messages and commit messages to the GLM. The disks are shared by every node.

The GLM has a role to perform the concurrency control and the cache coherency control. The *transaction scheduler* supports two-phase locking and deadlock resolution. The lock granularity is defined as a record. Transactions wait on any conflicting lock request, and are aborted only in case of deadlocks. Each deadlock is checked for each wait using the wait-for graph, and a transaction making the request is selected as a victim. Locks are released together at the end-of-transaction. The *cache manager* implements ARIES/SD algorithm [7], which is a representative cache coherency scheme in the SD cluster.

Table 2 shows the simulation parameters for specifying the resources and overheads of the system and the settings that will be used for experiments. Many of the parameter values are adopted from [2, 14].

We assume that the GLM's CPU performs much better than each node's CPU. The reason is to prevent the GLM from being the performance bottleneck;

hence, our experiment results could also be applied to the multi-GLM architecture. To apply the same rationale to the router, we also assume that the performance of router's CPU is equal to that of GLM's CPU. The number of shared disks are set to 10, and each disk has a FIFO queue of I/O requests. Disk access time is drawn from a uniform distribution between 10 milliseconds to 30 milliseconds.

We model that the database is logically partitioned into several *clusters*. Each database cluster has 10000 pages (40 Mbytes), and it is affiliated to a specific AC. The number of ACs is set to either 1 or 8. The transaction parameter of *ACLocality* determines the probability that a transaction operation accesses a data item in its affiliated database cluster. The *HotProb* parameter models "80-20 rule", where 80% of the references to the affiliated database cluster go to the 20% of the database cluster (*HotSize*). We refer the 20% of the database cluster as *hot set*, and the remaining part as *cold set*. The average number of records accessed by a transaction is determined by a uniform distribution between $TrxSize \pm TrxSize \times SizeDev$. The parameter *WriteOpPct* represents the probability of updating a record, and is set to 0.2. The processing associated with each record, *PerObjInst*, is assumed to be 15000 instructions.

The *network manager* is implemented as a FIFO server with 100 Mbps bandwidth. The CPU cost to send or to receive a message via the network is modeled as a fixed per-message instruction count plus an additional per-byte instruction increment. Note that the protocol processing (i.e., CPU overhead) dominates the on-the-wire time for messages in network.

The performance metric used in the experiments is the *throughput rate*. The throughput rate is measured as the number of transactions that commit per second. We also use an additional performance metric, *buffer hit ratio*, which gives the probability of finding the requested pages at local or remote buffers.

5 Experiment Results

In this section, our simulation model of SD cluster is used to explore the relative performance of routing algorithms. The compared algorithms are DACA, pure affinity-based routing (*PAR*), and dynamic affinity-based routing that spreads transactions of congested AC to all nodes in a round-robin fashion (*DRR*) [14]. In the following subsections, we first describe the performance characteristics of a single AC and then analyze the performance tradeoffs of three algorithms.

5.1 Performance of Single AC

To verify the correctness of simulation model, we explore the performance of a single AC by varying *NumNode*, *HotSize*, and *ClusterSize*. Figure 2 shows

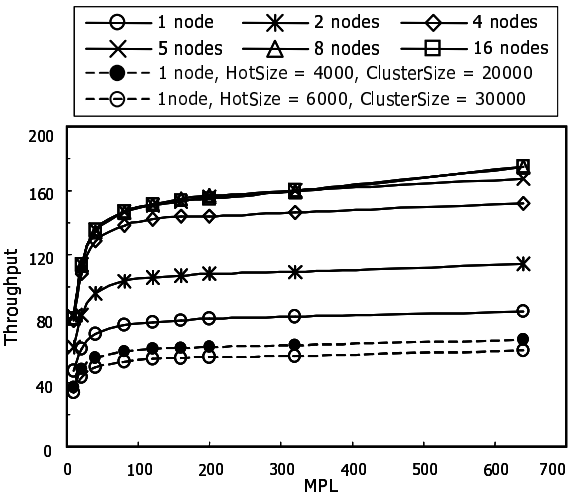

Figure 2: Transaction Throughput at Single AC

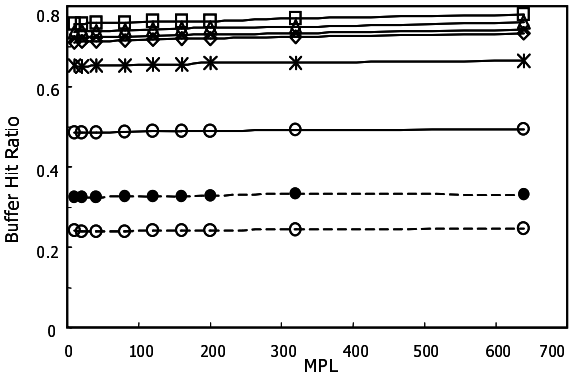

Figure 3: Buffer Hit Ratio at Single AC

the experiment results. The solid lines represent the transaction throughput when *HotSize* and *ClusterSize* are set to the values in Table 2, while *NumNode* is changed from 1 to 16. The transaction router allocates incoming transactions in a round-robin fashion among multiple nodes. The dashed lines show the performance when *HotSize* and *ClusterSize* increase as twice and three times respectively, while *NumNode* is 1. Note that it models the situation where multiple ACs are allocated to a node.

Allocating more nodes to an AC can exploit substantial performance improvement as expected. This is primarily due to the effect of increased buffer hit ratio as Figure 3 shows. When *NumNode* is 1 and *HotSize* is 2000, the buffer hit ratio is about 0.5. Since both *ACLocality* and *HotProb* are 0.8, the probability of accessing hot set is 0.64. Then the buffer hit ratio of 0.5 implies that part of hot set may not be cached in the buffer when *NumNode* is 1. This is not surprising even though *BufSize* is the double of *HotSize*. The probability of accessing other database clusters or cold set is rather high. As *NumNode* increases, most

pages in the hot set are cached at least one node's buffer, and thus the buffer hit ratio is over 0.64.

An interesting observation is that allocating extremely large nodes to an AC shows only marginal performance improvement. In our experiment, allocating over 4 nodes to an AC exhibits nearly similar performance behavior. The reason is when *NumNode* is over 4, most of a database cluster is cached, but all of database clusters cannot be cached even *NumNode* is 16. As a result, the buffer hit ratio is nearly equal at the area. Furthermore, the potential performance improvement due to large number of CPUs is offset by the increasing probability of buffer invalidation. This motivates our approach of DACA, where DACA allocates more node to an AC when it is *really* beneficial.

When *HotSize* and *ClusterSize* increase as twice and three times respectively, the performance goes down due to the lower buffer hit ratio. However, their performance differences are relatively small. This is because most of database accesses result in disk I/O.

5.2 Sudden Load Surge

We now compare the performance of three routing algorithms (DACA, PAR, DRR) in case of a load surge of a *single* AC. The extra load from the surge AC is not distributed at all in PAR, but DACA and DRR distribute the extra load to several nodes by the transaction router.

Figure 4 compares the throughput rate of three routing algorithms. Both *NumNode* and *ACNum* are set to 8, and *WriteOpPct* is 0.3. The value of *MPL* is 320, and thus the steady state load per each AC before the load surge is 40 transactions. The load surge of an AC is expressed as a fraction of its steady state load. For example, a load surge of 25% implies that the load of each non-surge AC decreases about 25% (10 transactions) and the total sum of additional load (70 transactions) goes to the single surge AC. As a result, the load of surge AC becomes 110 transactions, while 30 transactions are incoming per non-surge AC.

When the load serge is 0%, every algorithms perform in a similar manner. The load of each node is nearly equal in this area. As load surge increases, PAR performs worse as expected. Since PAR does not distribute the extra load from the surge AC to other nodes, PAR suffers from lower buffer hit ratio and limited computing facility for surge AC. Figure 5 shows that the buffer hit ratio of PAR is the lowest. As we have described in the previous experiment of single AC, part of hot set may not be cached in the buffer when a single node is allocated to an AC. This implies that the surge AC occurs tremendous buffer misses and disk I/Os in turn. When the load surge is 100%, the performance of PAR is nearly half compared to the other two algorithms. In this area, every executing transaction in the system corresponds to

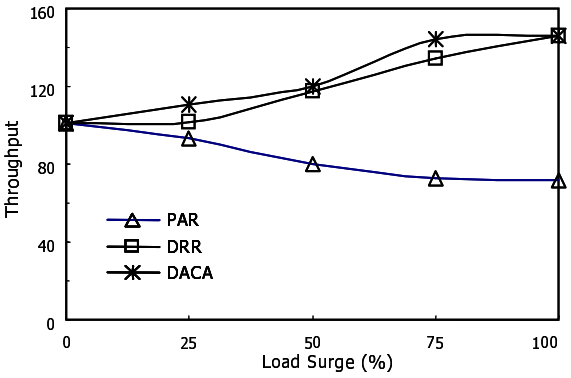

Figure 4: Transaction Throughput at Load Surge

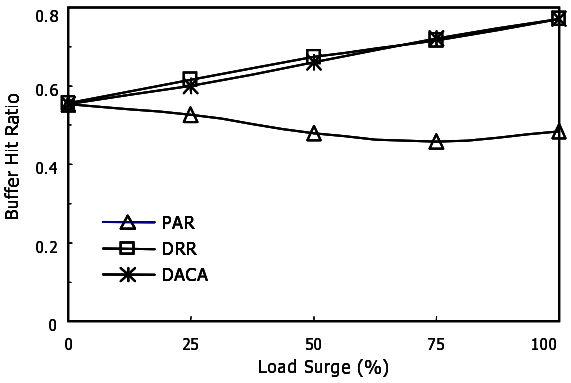

Figure 5: Buffer Hit Ratio at Load Surge

the surge AC, and the performance of PAR is equal to that of single AC with 1 node in Figure 2.

Both DACA and DRR performs better as the load surge increases. When the load serge is high, most transactions correspond to the surge AC and they are routed to several nodes. Allocating more nodes to the surge AC can achieve load balancing between nodes. Furthermore, the aggregate buffer space for the surge AC increases also. This is why the buffer hit ratio of both algorithms increases as the load surge increases.

DACA outperforms DRR when the load serge is between 25% and 75%. At first sight, this result might be inconsistent to the buffer hit ratio of Figure 5. The buffer hit ratio of DRR is rather slightly higher than that of DACA when the buffer hit ratio is 25% or 50%. However, it is important to note that the buffer hit ratio of Figure 5 adds both *local* buffer hit ratio and *remote* one. The local buffer hit ratio implies the probability of finding requested data from a node in its local buffer. On the other hand, the probability of finding the data from other node's buffer is represented as the remote buffer hit ratio. Figure 6 shows each ratio separately. Between the load serge of 25% and 75%, the local buffer hit ratio of DACA is higher than that of DRR. The reverse is true on the remote buffer hit ratio. High local buffer hit ratio is more

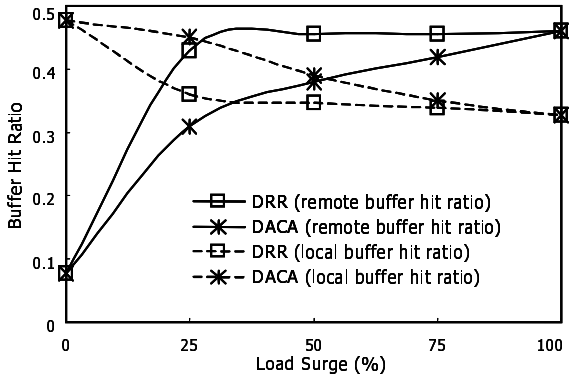

Figure 6: Local/Remote Buffer Hit Ratio at Load Surge

advantageous since the communication overhead due to page transfer between nodes can be reduced. DRR may reduce the amount of disk I/O by distributing the load of surge AC to every node, but it suffers from increasing communication overhead due to high remote buffer hit ratio. Furthermore, the performance improvement of DRR is marginal due to frequent buffer invalidation, which results in low local buffer hit ratio.

6 Concluding Remarks

Clustering multiple computing nodes for database transaction processing has received considerable interest in recent years for reasons of capacity, availability and cost. Furthermore, the rapid growth of cluster hardware technologies such as storage area networks enables to develop large scale database clusters. In this paper, we have proposed a dynamic transaction routing algorithm, named DACA, which selects a node for an incoming transactions so that the load of each computing node in the cluster can be evenly distributed. DACA is novel in the sense that it can achieve an optimal balance between affinity-based routing and dynamic load balancing.

We have compared the performance of DACA with other routing algorithms under a wide variety of database workloads and system configurations using an SD cluster simulation model. DACA outperforms the pure affinity-based routing algorithm significantly when transaction workload is changed dynamically due to sudden load surge. DACA also outperforms other affinity-based dynamic transaction routing algorithms by judicious node expansion strategy. The previous algorithms suffer from the overhead of frequent buffer invalidation and thus take a lower local buffer hit ratio. On the other hand, DACA can reduce the buffer invalidation effect by allocating more nodes to an AC when it is really beneficial.

References

[1] S. Chen, L. Xiao, and X. Zhang, "Dynamic Load Sharing with Unknown Memory Demands in Clusters," in *Proc. 21th Int. Conf. Distributed Computing Syst.* (2001) 109-118.

[2] H. Cho, "Cache Coherency and Concurrency Control in a Multisystem Data Sharing Environment," *IEICE Trans. Information and Syst.* E82-D(6) (1999) 1042-1050.

[3] H. Cho, "Database Recovery using Incomplete Page Versions in a Multisystem Data Sharing Environment," *Information Processing Letters* 83(1) (2002) 49-55.

[4] *DB2 Universal Database for OS/390 and z/OS - Data Sharing: Planning and Administration*, IBM SC26-9935-01 (2001).

[5] S. Haldar and D.K. Subramanian, "An Affinity-based Dynamic Load Balancing Protocol for Distributed Transaction Processing Systems," *Performance Evaluation* 17(1) (1993) 53-71.

[6] B. Hamidzadeh, L.Y. Kit, D.J. Lilja, "Dynamic Task Scheduling using Online Optimization," *IEEE Trans. Parallel and Distributed Syst.* 11(11) (2000) 1151-1163.

[7] C. Mohan and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," *Proc. 17th Int. Conf. on VLDB* (1991) 193-207.

[8] C.N. Nikolaou, M. Marazakis, and G. Georgiannakis, "Transaction Routing for Distributed OLTP Systems: Survey and Recent Results," *Info. Sciences* 97(1-2) (1997) 45-82.

[9] *Oracle9i Real Application Clusters - Concepts*, Oracle Part No. A89867-02 (2001).

[10] H. Schwetmann, *CSIM User's Guide for use with CSIM Revision 16*, MCC, (1992).

[11] B.A. Shirazi, A.R. Hurson, and K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press (1995).

[12] M. Yousif, "Shared-Storage Clusters," *Cluster Comp.* 2(4) (1999) 249-257.

[13] P. Yu and A. Dan, "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," *IEEE Trans. Knowledge and Data Eng.* 6(5) (1994) 764-786.

[14] P. Yu and A. Dan, "Performance Evaluation of Transaction Processing Coupling Architectures for Handling System Dynamics," *IEEE Trans. Parallel and Distributed Syst.* 5(2) (1994) 139-153.