Performance Analysis of Coherency Control Policies through Lock Retention

Asit Dan and Philip S. Yu
IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

Buffer coherency control can be achieved through retaining a lock (shared, exclusive, etc.) on each page in the buffer, even after the requesting transaction has committed. Depending upon the lock mode held for retention and the compatibility of lock modes specified, different retention policies can be devised. In addition to tracking the validity of the buffered data granules, additional capabilities can be provided such as deferred writes to support no-force policy on commit, (node) location identification of valid granules to support remote memory accesses, and shared/exclusive lock retention to reduce the number of global lock requests for concurrency control. However, these can have serious implications not only on the performance but also on the recovery complexity. In this paper, five different integrated coherency policies are considered. We classify these policies into three different categories according to their recovery requirements. A performance study based on analytic models is provided to understand the trade-offs on both maximum throughputs and response times of the policies with a similar level of recovery complexity and the performance gain achievable through increasing the level of recovery complexity.

1 Introduction

The coherency problem of software managed buffers is common to several multi-system environments where multiple computing systems share a common set of data pages and each system (also referred to as node) caches recently accessed data pages in its local buffer to reduce the number of I/O operations or remote data accesses. Examples are transaction processing environments where multiple nodes share a common database [18, 10, 21, 15, 4, 12], client-server environments where client nodes may access the same set of files stored in the server [9, 20, 3, 23] and the distributed shared memory environments where physical pages corresponding to a virtual page may be present at multiple computing nodes [11, 14]. In this paper, we focus on the transaction processing application in a multi-system data sharing environment.

1992 ACM SIGMOD - 6/92/CA, USA

In [6] six different coherency policies are examined and broadly classified into three categories: *detection* of invalidated pages, *notification* of invalidated pages and *propagation* of updated pages. Under these policies, at the transaction commit point the updates are propagated to the disk (called the *force* scheme) and the locks are then released. Several policies considered in the simulation studies in [20, 3, 23] for a high data contention client-server environment deal with efficient resolution of conflict across the active transactions in the system. Most of these policies also fall under the category of the force scheme with no lock retention. Alternatively, *physical* and *logical* locks can be obtained to maintain coherency and concurrency, respectively [13]. The two types of locks can be combined into a single lock (referred to as LP lock in [13]) to reduce message overhead under integrated concurrency and coherency control policies [2, 8, 15, 16, 13, 14]. Therefore, the coherency is maintained by retaining a LP lock on each buffer page in a node even after the transaction accessing the page is committed so that updates to these pages in another node are prohibited until these locks are revoked [16, 12, 9, 23].

Depending upon the lock mode held for retention and the lock mode compatibility specified, different retention schemes can be devised. In addition to tracking the validity of the buffered pages, additional capabilities can be provided such as deferred writes to support the *no-force* scheme at commit [12], (node) location identification of valid granules to support remote memory accesses, restriction on the number of replications across buffers, and shared/exclusive lock retention to reduce the number of global lock requests for concurrency control. In [15, 23], a shared lock retention approach (called *read optimization* scheme) is used to provide coherency control under the *force* scheme. The approach saves the global lock request message for a read access if the lock is already retained in the executing node. Retention of exclusive lock (also known as *sole interest*) is also proposed in [16]. Mohan and Narang, [12], have proposed and detailed several buffer coherency policies based on exclusive lock retention to support a *no-force* scheme at commit. These policies differ at the timing when an updated page is forced out to disk and the way remote memory transfer is carried out. The ways by which the writes are deferred can have serious implications to *recovery complexity* in terms of processing overhead and recovery time. It is assumed that each node maintains a separate log. Deferred writes within

a single node require going back further into the log of that node to redo the missing updates during recovery. This is referred to as *medium* recovery characteristic in contrast to the *simple recovery* characteristic of the policies under the *force* scheme. Deferred writes across multiple nodes, i.e. letting a page containing updates from multiple nodes before writing back to disk, would require a log merge across nodes before recovery can be started. This is referred to as *complex recovery* characteristic.

In this paper, we consider five different integrated concurrency coherency policies under the three recovery characteristics mentioned above, and develop analytical models to study their relative performance. Two of the policies are variants of the schemes considered in [12] with medium and complex recovery characteristics, respectively. We propose additional variations in policies based on single copy constraint, i.e. no page is allowed to be buffered by more than one node. Without any loss of generality, we consider this variation only for the policies with complex recovery characteristic. For the policies with simple recovery characteristic, besides the check-on-access policy [2, 8, 15] (detailed later) which serves as our base policy, we also consider a shared lock retention policy under the force scheme [15, 23]. All policies except the basic check-on-access policy exploit direct memory to memory transfer. Apart from the differences in recovery complexity, there are obvious performance trade-offs among these policies. While increasing the recovery complexity, the deferred write or no-force scheme obviously reduces the number of disk writes. But additional number of messages are incurred to maintain coherency when a dirty page, i.e. an updated page not yet written to disk, is needed by another node. Details of these policies and their performance trade-offs are discussed in Section 2.

The paper is organized as follows. Section 2 describes the details of the the five buffer coherency policies that are considered in this paper. The analytical models to estimate the local and remote buffer hit probabilities, the CPU overhead due to various types of coherency messages and the overall transaction response time are developed in Section 3. The performance comparisons are presented in Section 4. A summary and concluding remarks appear in Section 5.

2 Buffer Coherency Policies

The multi-system data sharing environment considered here consists of multiple loosely coupled nodes sharing a common database at the disk level. We assume the presence of a fast speed interconnect amongst the nodes, such that propagation delay (not the CPU overhead) in transferring a page between two nodes is negligible.[1] Standard two phase locking is used for concurrency control where a Centralized Global Lock Manager (GLM) is assumed to be available which can be implemented either on control units [1] or through some specialized processor [17]. The access conflict across transactions executing in different nodes is resolved by the GLM while the access conflict across transactions executing in the same node is resolved by the Local Lock Manager (LLM) in each node. Each node maintains a local buffer under the LRU buffer replacement scheme to cache a part of the database. The LLM retains a LP lock associated with each buffer page after the requesting transaction has committed if there is no transaction waiting for this lock in another node. A retained LP lock is released or downgraded, and the associated buffer copy may be purged (depending on the lock conflict) if a lock on the same page is subsequently requested by another transaction executing in a different node. The retained LP lock is also released if the buffer copy is pushed out of the buffer. Note that the LP lock can be either shared(S) or exclusive(X). A shared LP lock implies that only the shared read requests can be granted to a local transaction, and the page can be present in other buffer. On the other hand, an exclusive LP lock ensures that no other node has a copy of this page and update requests can be granted locally. Additional lock mode may be required for some of the policies, as described below.

At transaction commit time, *write-ahead logs* are forced to the disk to guarantee that the updates can be permanently reflected onto the database. However, the updated pages need not be immediately propagated to the disk, if exclusive LP locks on the updated pages are retained by the LLM.[2] This is referred to as deferred write [19]. If the updated pages are further updated by any subsequent transactions in the same node, only the final versions need to be propagated to the disk before releasing the exclusive locks. This way many write I/Os can be saved, improving system throughput and response time. However, the deferred write scheme increases the recovery complexity, as the updates in the write-ahead logs that have to be examined and applied may go back for a long period of time in case of system failure.

A retained X lock needs to be either downgraded to an S lock or released, if a lock on the same page is requested by a remote node either in S or X mode. To keep the recovery scheme simpler in the medium recovery scheme, the updated page needs to be propagated to the disk before the X lock is downgraded or released. However, this results in additional I/O delay and larger transaction response time. A copy of the updated page is transferred directly to the requesting node, after the propagation to the disk is completed. This type of schemes with *medium* recovery complexity is referred to as the *Medium* scheme in [12] due to performance consideration. (Note that in [12], the logical and physical locks are to different entities (page and record) and hence are separate. The focus is on data or physical lock retention. Here logical and physical locks are combined and both physical and logical lock retentions are considered [13].)

The write I/O can be further deferred (and perhaps saved) at the expense of increasing recovery complexity by not propagating the updated pages to the disk upon transfer. Since the deferred write now involves updates by transactions across multiple nodes, a more complex recovery scheme

[1] With a small modification, the model can also be used for the environments where the network delay is not negligible, like the Client-Server environment connected through the local area network. The qualitative results will still be the same. However, the response time gap between the SRS and DF policies (detailed later) will be reduced.

[2] Note that locks can not be downgraded or released by the node until the pages are propagated to the disk to guarantee that other nodes see the most recent copy of these pages.

that uses merged logs of all nodes is required. This type of schemes with *complex* recovery is referred to as the *Fast* scheme in [12] for its ability to have faster page transfer and the most reduction in write I/O. (Recovery issues corresponding to the fast and medium schemes are discussed in detail in [12].)

Within each level of recovery complexity, there are additional dimensions that result in many different coherency control policies. For example, in [6] six different coherency policies are evaluated which are all with *Simple* recovery characteristic and no logical lock retention. The policies trade off response time and throughput in different ways. The Check-on-Access policy was found to provide shorter response time as well as higher throughput for a wide range of system and workload parameters. Here, our focus is on *data and lock* retention. Apart from the recovery aspect, a new dimension considered here is to restrict the number of in-buffer replications to fully exploit the advantage of remote memory accesses. This can improve the buffering efficiency, and hence, reduce the number of disk reads, but increase the local buffer misses, and hence, increase the number of messages, and memory-to-memory transfers. Even without resorting to deferred writes to keep recovery simple, S lock retention on a previously accessed page can be used to save the number of lock requests if subsequent accesses to that page are from the same node. Otherwise, additional messages will incur. Thus the number of messages needed to resolve the coherency and serialization requirements and the number of I/Os are quite different for these schemes. Therefore, in addition to the check-on-access policy, we select four alternative policies (choice of them should be clear, as the results are presented) and evaluate their relative performance through analytical models.

2.1 Schemes with Simple Recovery

As mentioned earlier, under this recovery scheme writes are forced at the transaction commit time. However, the propagation delay of write I/Os is not included in the transaction response time. We choose two different policies under this recovery scheme: one retains no LP lock while the other retains only the shared LP lock.

Check-on-Access(CA) Policy: Under this policy, the obsolete pages are detected at the page access time by a transaction. At the transaction commit time, all LP locks are downgraded to weak locks so that the GLM can track the nodes buffering each page using a valid bit per node and mark their copies invalid upon subsequent updates by another node [8]. Before accessing any page, the processing node of the transaction makes a global lock request to the GLM. In response, the integrated GLM returns not only the requested lock, but also the result of the associated buffer validity check based on the valid bit. Note that the page is valid at the lock downgrade time at the executing node. At the lock downgrade time, if the page has been updated, the valid bits corresponding to all other weak lock holders except the updating node (which is the only node with an up-to-date version of that page) will be turned off. The policy certainly saves the overhead of sending immediate notification of page invalidations to other nodes, but it also reduces the buffer hit probability as the obsolete pages continue to reside in the local buffer.

Short term Retention of Shared Lock (SRS) Policy: Here, the S locks are retained by the node even after the transaction has completed, but until the associated data page is flushed out of buffer. The X locks are downgraded to S locks, so that an S lock request by a remote node can be granted by the GLM immediately. Retention of S locks has two advantages. First, all S lock requests by subsequent local transactions can be granted without the overhead of GLM lock requests, if they have been retained. Otherwise, lock requests are made to the GLM. Secondly, through the retained S locks, the GLM can identify the nodes with a valid buffer copy, and notify it to forward a copy to the requesting node. However, if an X lock is requested by a remote node GLM has to revoke all the S locks (also the pages are purged from buffers) before the X lock is granted. This incurs a large message overhead, negating the advantage of retention.

2.2 Schemes with Medium Recovery

Deferred until Transfer or Flushing (DTF) Policy: Here, the writes (or dirty pages) are deferred as long as the X locks are retained. The writes are forced to the disk before a node to node page transfer occurs due to a lock request by a remote node either in an S or X mode. If the remote request is for an S lock then the local X lock is downgraded to S mode. Otherwise, if the remote request is for an X lock, the local X lock is released and the local buffer copy is purged. The writes are also propagated to the disk if an updated page is pushed out of buffer, referred to as buffer flushing. When a page is pushed out of the buffer, its associated lock (S or X) is also released. In actual implementation, the write propagation can be initiated once the updated page is close to the bottom of the LRU stack, so that no synchronous I/O delay is incurred due to buffer flushing.

2.3 Schemes with Complex Recovery

Deferred until Flushing (DF) Policy: This is similar to DTF policy as far as propagation of updated pages to the disk at the buffer flushing goes. However, on a lock request by a remote node, updates are not propagated to the disk. If the remote request is for an X lock, like the DTF policy both the updated page as well as the X lock is transferred to the requesting node. On the other hand, if the remote request is for an S lock, the local node downgrades its X lock to the shared mode, and a copy of the page is also sent to the requesting node. Note, that updated page still needs to be propagated to the disk by one of the nodes eventually. Therefore, a new form of shared lock called U lock is introduced [12]. The local shared lock is in U mode implying a pending update. Therefore, at the buffer flushing time, not only the pages associated with X locks but also the ones with U locks are propagated to the disk.

One copy along with Deferred until Flushing (ODF) Policy: Both DTF and DF policies can have a variant, in which only one node at a time is allowed to retain an LP lock. This can improve the buffer efficiency by disallowing replications in multiple buffers. However, the policy

can be inefficient if there is enough buffer available in the system, and replication of hot data can reduce substantially node to node page transfer. Since, this is a separate dimension, we consider this policy only for the case with complex recovery, without loss of generality.

3 Performance Model

The transaction response time can be broken down into three parts: 1) synchronous I/O delay (due to read or write of data pages and log write), 2) CPU queueing delay and service time for application processing and for synchronous protocol overhead due to concurrency and coherency, and 3) waiting time due to lock contention. The synchronous I/O delay depends on the the private and remote buffer hit probabilities that determine the number of synchronous read and write I/O operations to be performed by a transaction. (Note that under the DTF policy write propagation due to read or write on a remote dirty page is also synchronous.) The CPU queueing delay and service time depends not only on the total number of CPU instructions executed synchronously, I_{syn}, as a part of transaction response time but also on the CPU utilization level which in turn depends on both I_{syn} and I_{asyn}. Here, I_{asyn} is the number of CPU instructions executed asynchronously per transaction due to write propagation. We will use a hierarchical modeling approach where the buffer hit probabilities, concurrency control and system resource access times are modeled separately and the interactions amongst the submodels are captured via a higher level model [4, 5, 6]. Due to space limitation, we will omit the details of concurrency control model that can be found in [22]. However, we will provide the details of the buffer model and the estimation of concurrency coherency message overhead that is used to determine I_{syn} and I_{asyn}. Further details on the analysis can be found in [7].

We first develop the buffer model under the LRU replacement policy that provides the estimation of various types of LP locks (S, X, U locks) retained in a particular buffer. We will then present the estimation of local and remote buffer hit probabilities and the estimation of I_{syn} and I_{asyn}. The data sharing system considered here consists of N loosely coupled nodes. Each node maintains a local buffer of size B. Transactions arrive at each node according to a Poisson process with rate λ and each transaction accesses L pages from the shared database. The access pattern over the entire database is not the same for all transactions. First, we assume the related transactions (i.e., transactions that reference the same set of relations) can be grouped logically into *affinity clusters* (AC) [22] and the transactions associated with an affinity cluster are routed to the same node by the front end router. This reduces the inter-system interference and improve the local buffer hit probability. The relations associated with each AC is referred to as a DB cluster. Let η be the fraction of pages accessed by a transaction from its associated DB cluster. We will assume each transaction accesses the remaining fraction, $(1-\eta)$, of its data from the non-associated DB clusters uniformly. Let each DB cluster consists of D pages. Second, we assume the database accesses within each DB cluster is skewed. We model skewed access pattern by assuming P logical partitions within each cluster such that the probability of accessing any page within a partition is uniform. Let β_p denote the fraction of the DB cluster pages in partition p, i.e., the size of partition p is $\beta_p D$. Let α_p be the probability of accessing a page of the p^{th} partition of this cluster, given that a page is accessed from this DB cluster. The probability that an accessed page is also updated is denoted as γ.

3.1 Buffer Model

We now focus our attention to the buffer at the m^{th} node. The parameter m is fixed for the following set of equations, and the analysis can be repeated for all other nodes. For the purpose of analysis, the entire database can be divided into NP partitions, i.e., N DB clusters and P partitions within each cluster (based on the hotness of the data). Each partition is identified by the subscript pair $(n,p), n \in \{1...N\}, p \in \{1...P\}$. Define $D_{n,p}$ to be the size of partition (n,p). Then, $D_{n,p}$ is given by $D\beta_p$. Note that transactions at node m have higher affinity to the partitions (m,p), $p = 1...P$. We will develop the model for the DF policy (which is the most complicated case). The analysis can be easily adapted for the other cases. To estimate the steady state probability of buffer hit, we first derive the average number of pages of each partition in the local buffer holding S, X or U locks. Our analysis extends the methodology used in [5, 6] for estimating the buffer hit probability under the broadcast invalidation or check-on-access policy. Let $Y_{n,p,m}(k)$ denote the average number of pages of partition (n,p) present in the top k buffer locations of the m^{th} node. Similarly, $S_{n,p,m}(k)$, $X_{n,p,m}(k)$ and $U_{n,p,m}(k)$ denote the average number of pages holding S, X and U locks, respectively, in the top k locations of the LRU stack at node m. Let $\lambda^l_{n,p,m}$ be the rate at which data of partition (n,p) are accessed by the local transactions of node m, and $\lambda^r_{n,p,m}$ be the rate at which they are accessed by the remote transactions. Now, for $p = 1, ..., P$,

$$\lambda^l_{n,p,m} = \begin{cases} \lambda L \alpha_p \eta, & \text{if } n = m \\ \lambda L \alpha_p \frac{(1-\eta)}{(N-1)}, & \text{otherwise,} \end{cases}$$

$$\text{and} \quad \lambda^r_{n,p,m} = \begin{cases} \lambda L \alpha_p (N-1) \frac{(1-\eta)}{(N-1)}, & \text{if } n = m \\ \lambda L \alpha_p \left\{ \eta + \frac{(N-2)(1-\eta)}{(N-1)} \right\} & \text{otherwise.} \end{cases}$$

Let $y_{n,p,m}(k)$ be the probability that the k^{th} buffer location from the top of the LRU stack at node m contains a page of partition (n,p). Let $x_{n,p,m}(k)$ and $u_{n,p,m}(k)$ be the probabilities that the page is of partition (n,p) and is holding an X and U lock, respectively. Then,

$$Y_{n,p,m}(k) = \sum_{l=1}^{k} y_{n,p,m}(l), \quad X_{n,p,m}(k) = \sum_{l=1}^{k} x_{n,p,m}(l),$$

$$U_{n,p,m}(k) = \sum_{l=1}^{k} u_{n,p,m}(l), \qquad \text{and}$$

$$S_{n,p,m}(k) = Y_{n,p,m}(k) - (X_{n,p,m}(k) + U_{n,p,m}(k)).$$

We will set up a recursive formulation to determine $y_{n,p,m}(k+1)$, $x_{n,p,m}(k+1)$ and $u_{n,p,m}(k+1)$ for $k \geq 1$ given $y_{n,p,m}(l)$, $x_{n,p,m}(l)$ and $u_{n,p,m}(l)$, for $l = 1, ..., k$. Consider a smaller buffer consisting of the top k locations only. The buffer location $(k+1)$ receives the page that is pushed

down from location k. Let $r_{n,p,m}(k)$ be the rate at which pages of partition (n,p) are pushed down from location k. Similarly, let $r^X_{n,p,m}(k)$ and $r^U_{n,p,m}(k)$ be the rates at which pages of partition (n,p) holding X and U locks, respectively, are pushed down from location k. Our estimation of $y_{n,p,m}(k+1)$, $x_{n,p,m}(k+1)$ and $u_{n,p,m}(k+1)$ is based on the following: 1) the relative push down rate for each data type from location k is approximated as the expected value of finding a page of the $(n,p)^{th}$ partition in the $(k+1)^{st}$ buffer location over all time, and 2) push down rate for each data type at location k can be derived using the conservation of flow argument for each data type separately. (See, [4, 5, 7] for more details on the methodology.) Therefore, using the conservation of flow for the pages holding any lock or, X or U locks, we equate the long term rate at which these pages of the $(n,p)^{th}$ partition get pushed down from the top k locations of the buffer, as

$$r_{n,p,m}(k) = \lambda^l_{n,p,m}\left[1 - \frac{Y_{n,p,m}(k)}{D_{n,p}}\right] - \lambda^r_{n,p,m}\gamma\frac{Y_{n,p,m}(k)}{D_{n,p}},$$

$$r^X_{n,p,m}(k) = \lambda^l_{n,p,m}\gamma\left[1 - \frac{X_{n,p,m}(k)}{D_{n,p}}\right] - \lambda^r_{n,p,m}\frac{X_{n,p,m}(k)}{D_{n,p}}$$

$$r^U_{n,p,m}(k) = \lambda^r_{n,p,m}(1-\gamma)\frac{X_{n,p,m}(k)}{D_{n,p}} - \lambda^r_{n,p,m}\gamma\frac{U_{n,p,m}(k)}{D_{n,p}}$$

The probability $y_{n,p,m}(k+1)$, can be approximated as

$$y_{n,p,m}(k+1) \approx \frac{r_{n,p,m}(k)}{\sum_{i\in\{1..N\},j\in\{1...P\}} r_{i,j,m}(k)}.$$

The probabilities that the page at the $(k+1)^{st}$ location is of partition (n,p), and it holds an X and U locks, respectively, are given by their joint probabilities. Therefore,

$$x_{n,p,m}(k+1) \approx \frac{r^X_{n,p,m}(k)}{r_{n,p,m}(k)} y_{n,p,m}(k+1),$$

$$u_{n,p,m}(k+1) \approx \frac{r^U_{n,p,m}(k)}{r_{n,p,m}(k)} y_{n,p,m}(k+1).$$

The above equations are solved iteratively, with the base conditions of $y_{n,p,m}(0) = 0$, $x_{n,p,m}(0) = 0$, and $u_{n,p,m}(0) = 0$. At the point, when $Y_{n,p,m}(k)$ is very close to its limit $D_{n,p}$, $Y_{n,p,m}(k)$ may exceed $D_{n,p}$ because of the approximation in the above equations. This is corrected by resetting $Y_{n,p,m}(k)$ to $D_{n,p}$ whenever $Y_{n,p,m}(k)$ exceeds $D_{n,p}$ and $r_{n,p,m}(k)$, $r^X_{n,p,m}(k)$ and $r^U_{n,p,m}(k)$ are taken to be zero for all subsequent steps for that partition. The above approximations are very accurate and particularly for a large buffer size. Similar approximations in [5, 6] showed excellent agreement. More discussions on the accuracy can be found in [4, 5].

3.2 Estimation of Local and Remote Buffer hit probabilities

We now show how to estimate the local, remote and overall buffer hit probabilities from the point of view of the transactions at the m^{th} node given the composition of buffers at all nodes. These probabilities provide not only the estimates of the average number of read I/O operations saved, but also the average number of messages of various types incurred (depending on the policy), and the average number of write propagations to disks. We will first focus our analysis on the DF policy, and then will discuss the modifications needed for the other policies. The U locks are different from the S locks only from the point of view of write propagation at the time the page is pushed out of the buffer. Hence, from the transaction lock conflict resolution point of view (i.e., for the average number of messages needed to revoke all shared locks, etc.) the same actions are taken for both S and U locks. Note that the sets of pages holding X locks in different buffers are mutually disjoint, while more than one copy of other pages may be present in different buffers. Therefore, the computation of remote buffer hit probabilities for the pages with X locks is different from those of the pages with S or U locks.

Let $t_{n,p,m}$ be the average number of pages accessed by a transaction at the m^{th} node from the partition (n,p). Then $t_{n,p,m}$ is given by $t_{n,p,m} = \lambda^l_{n,p,m}/\lambda$. Given that the next page accessed by a transaction at node m lies in partition (n,p), we need to estimate the following probabilities:
1) the probability that the page is found in the local buffer of the m^{th} node holding an X lock, $h^{l,X}_{n,p,m}$,
2) the probability that the page is found in one of the remote buffers holding an X lock, $h^{r,X}_{n,p,m}$,
3) the probability that the page is found in the local buffer of the m^{th} node holding an S or U lock, $h^{l,SU}_{n,p,m}$, and
4) the joint probability that the page is found in one of the remote buffers holding an S or U lock, and it is not found locally, $h^{r,SU}_{n,p,m}$.
The probabilities, $h^{l,X}_{n,p,m}$, and $h^{r,X}_{n,p,m}$ can be estimated in a straightforward way, since the events are mutually disjoint. Therefore,

$$h^{l,X}_{n,p,m} = \frac{X_{n,p,m}(B)}{D_{n,p}}, \quad h^{r,X}_{n,p,m} = \frac{\sum_{i=1,i\neq m}^{N} X_{n,p,i}(B)}{D_{n,p}}.$$

The pages with shared locks can be present in multiple buffers, but can not lie in the set of pages holding X locks. Hence,

$$h^{l,SU}_{n,p,m} = \left[1 - \left\{h^{l,X}_{n,p,m} + h^{r,X}_{n,p,m}\right\}\right] \left[\frac{\{Y_{n,p,m} - X_{n,p,m}\}}{\left\{D_{n,p} - \sum_{i=1}^{N} X_{n,p,i}(B)\right\}}\right].$$

The second term in the above equation is the conditional probability that the page is found locally given that it does not lie in the set of pages holding an X lock. Similarly,

$$h^{r,SU}_{n,p,m} = \left[1 - \left\{h^{l,X}_{n,p,m} + h^{r,X}_{n,p,m}\right\}\right] \left[1 - \frac{\{Y_{n,p,m} - X_{n,p,m}\}}{\left\{D_{n,p} - \sum_{i=1}^{N} X_{n,p,i}(B)\right\}}\right] \left[1 - \prod_{j=1,j\neq m}^{N}\left\{1 - \frac{\{Y_{n,p,j} - X_{n,p,j}\}}{\left\{D_{n,p} - \sum_{i=1}^{N} X_{n,p,i}(B)\right\}}\right\}\right].$$

As before, the first term is the probability that the page does not lie in the set of pages holding an X lock. Given that the page does not lie in the set of pages holding an X lock, it can still appear in both local and remote buffers independently. The second term is the conditional probability that the page is not present in the local buffer given the first condition. And the third term is the conditional probability that the page appears in one of the remote buffers given the first condition (the second condition has no effect on this term since the events are independent).

One more related entity is the average number of shared (S or U) locks revoked per access, $N^{r,SU}_{n,p,m}$. This will happen

if the next access is an update and the page does not lie in the set of pages holding a X lock. Therefore,

$$N^{r,SU}_{n,p,m} = \gamma\left[1-\left\{h^{l,X}_{n,p,m}+h^{r,X}_{n,p,m}\right\}\right]\left[\sum_{j=1,j\neq m}^{N}\frac{\{Y_{n,p,j}-X_{n,p,j}\}}{\left\{D_{n,p}-\sum_{i=1}^{N}X_{n,p,i}(B)\right\}}\right].$$

The above equations for the local and remote buffer hit probabilities and for the average number of shared lock revocation messages also hold for the DTF and SRS policies. As mentioned earlier, no X locks are retained under the SRS policy, however, the interpretation in the above analysis of the pages with X locks is that those pages hold S locks under the SRS policy but are recently updated and not yet replicated in other buffers.

Under ODF policy, each page can be present in at most one buffer, irrespective of the lock mode retained on that page. Therefore, all events are independent and the equations for $h^{l,SU}_{n,p,m}$, and $h^{r,SU}_{n,p,m}$ are modified as

$$h^{l,SU}_{n,p,m} = \frac{\{Y_{n,p,m}-X_{n,p,m}\}}{D_{n,p}},$$

$$h^{r,SU}_{n,p,m} = \sum_{i=1,i\neq m}^{N}\frac{\{Y_{n,p,i}-X_{n,p,i}\}}{D_{n,p}}.$$

Finally, the overall buffer hit probability for a page of partition (n,p) for all policies (except the CA policy which does not make use of remote buffers), $h_{n,p,m}$, can be written as $h_{n,p,m} = \left(h^{l,X}_{n,p,m}+h^{r,X}_{n,p,m}+h^{l,SU}_{n,p,m}+h^{r,SU}_{n,p,m}\right)$.

3.2.1 Estimation of CPU Overhead

We will show the estimation of I_{syn} and I_{asyn} for the DTF policy since this part of analysis is most complex for the DTF policy. The estimation for the other policies can be done in a very similar way. Let I_{trx} be the base number of instructions executed per transaction excluding the I/O and coherency overhead. Let I_{io} be the number of CPU instructions required per disk I/O operation. Also, let I_{GLM} be the number of instructions executed by a node per communication (e.g., lock request) with the GLM and I_{page} be the total number of instructions executed (combined overhead of the sender and receiver nodes) to transfer a page between two nodes.

The next page to be accessed by a transaction may be in one of the five states (4 buffer states as enumerated in the earlier subsection and the buffer miss state), and different actions are taken depending on the state. The action also depends on the access mode of the current transaction. The actions are enumerated as follows:

1. **local exclusive:** No action is needed for this state as both the highest mode of LP lock and up-to-date data are retained.
2. **remote exclusive:** Local node requests the GLM for an S or X depending on the access mode. The GLM requests the node holding the X lock to downgrade it to S mode or revoke the X lock depending on the lock request mode. After receiving the request from the GLM, the remote node first propagates the dirty page to the disk and then sends a copy directly to the requesting node. It then downgrades or releases its lock as per request, and subsequently, the GLM grants the requested lock to the local node.
3. **local shared:** No action is needed for an S lock request, but all remote S lock revocations through the GLM are required if the next access is in X mode.
4. **remote shared:** Request for an S lock is granted by the GLM immediately. The GLM also requests one of the remote S lock holding nodes to transfer a page to the requesting node. In case of an X lock request, all the remote S locks are revoked before the X lock is granted by the GLM, and one of the S lock holding node also transfers a copy of that page. Note that the request for page transfer and the lock revocation can be combined into a single message by the GLM.
5. **buffer miss:** The lock request is granted by the GLM immediately as no node is holding a lock. The page is read from the disk.

The I_{syn} can now be expressed as,

$$I_{syn} = I_{trx} + \sum_{n\in\{1\ldots N\},p\in\{1\ldots P\}} t_{n,p,m}\{ h^{r,X}_{n,p,m}(4I_{GLM}+I_{page}+I_{io}) + N^{r,SU}_{n,p,m}2I_{GLM} + h^{l,SU}_{n,p,m}\gamma 2I_{GLM} + h^{r,SU}_{n,p,m}(2I_{GLM}+I_{page}) + (1-h_{n,p,m})(I_{GLM}+I_{io})\}.$$

Here, the first term is for the remote exclusive case, and it involves four communications to the GLM (2 on the requester side and 2 on the remote node), a page transfer and a disk I/O propagation. The second term includes all revocations of remote shared locks (2 per lock holding node). The third term is for the local shared case, and it includes only the request for upgrade to X mode, since the remote S lock revocation is already included in the second term. The remaining two terms are for the remote shared and buffer miss cases.

I_{asyn} involves only the propagation of deferred writes at the buffer flushing time. Note that the release of associated lock can be piggybacked with the next communication with the GLM, and therefore, incurs no extra overhead. The additional CPU overhead per transaction, I_{asyn}, can be written as,

$$I_{asyn} = \sum_{n\in\{1\ldots N\},p\in\{1\ldots P\}} r^{X}_{n,p,m}(B)I_{io}/\lambda.$$

Recall from earlier subsection that the $r^{X}_{n,p,m}(B)$ is the rate of flushing of the dirty pages from the buffer. For the DF policy the numerator also includes additional terms capturing the flushing of pages with U lock.

4 Performance Comparisons

We will first compare the performance of the CA, DTF and DF policies, which are the base policies for each category of recovery characteristic. We will then consider the variations within each recovery category to understand the impact of other dimensions. We assume a transaction profile similar to that of a relational transaction workload based on the inventory tracking and stock control application used in [21, 22]. The transaction has a pathlength of 250K instructions (I_{trx})

and requests 14 locks. The overhead for communication with the GLM (I_{GLM}) is assumed to be 2K instructions while the overheads for transferring a page (I_{page}) and disk I/O (I_{io}) are both taken to be 5K instructions. The coupled system or cluster consists of 8 nodes where each node has a 20 MIPS processor. The I/O time for read or write is 25 milliseconds while that for writing the log is 5 milliseconds. Each node has a 25K page (100 Mbytes) local buffer (unless otherwise specified). The database consists of 50K granules (200 Mbytes) per node (D_p). The access pattern to each database cluster is assumed to follow the 80-20 rule ($P = 2$). The transaction and system parameters are kept fixed for all studies, unless otherwise specified.

Comparison of the CA, DTF and DF Policies: Figure 1 shows the effect of transaction node affinity on the response time of the three policies. The CA policy has the worst response time as it does not take advantage of the remote buffers. The problem is most severe for the case of lower affinity as the local buffer hit probability becomes very small. Both DTF and DF improve their response times by accessing a page directly from a remote buffer if it is not found locally. In the case of DTF policy, if the remote page is updated, then the page is first propagated to the disk before the X lock is downgraded. This extra I/O delay is eliminated for the DF policy. Also shown is a set of curves (dashed lines) corresponding to a higher update probability, ($\gamma = 0.5$). The small amount of further increase in response time under the CA policy is due to further drop in local buffer hit probability. However, the increase in response time for the DTF policy is quite significant (response time is close to that of CA policy) as a larger number of remote pages are held in X mode. Note that for high affinity the performance of all three policies are very close.

Figure 2 shows the CPU utilizations corresponding to the case in Figure 1. For lower transaction affinity, both the DTF and DF policies incur more message overhead and therefore, CPU utilizations are higher. The utilizations of both the DTF and DF policies are close for a lower update probability (0.2). However, the gap in utilization between the two policies increases for a higher update probability (0.5). The difference is due to the higher increase in the number of write I/O propagations for the DTF policy. The overall trade-offs of response times and maximum transaction throughputs are shown in Figure 3 for higher and lower affinity values (0.9,0.4). For higher affinity, both the DTF and DF policies can provide not only lower response time but also higher maximum throughput as there is very little remote message overhead. With lower transaction affinity, the local buffer hit probabilities decrease for all policies. The DF policy can still keep its response time lower for lower CPU utilizations by accessing remote buffers, however incurring significant message overhead. The DTF policy loses some of the advantage of remote buffer accesses as write propagation delay is incurred for the remote dirty pages.

Since the difference in both response time and maximum throughput between the DTF and DF policies depends on the number of write I/O operations, the savings in write I/O due to deferred write (and hence, rewrite) are shown in Figure 4 as a percentage of the maximum write I/O operations (i.e., write I/Os under the force scheme). The differences between the solid and the corresponding dashed lines are the amounts of forced I/O due to remote accesses under the DTF policy. Write I/O savings increase with the increase in buffer size. The savings are much higher for higher affinity (marked with 'H'), as very few updated pages are flushed or forced. Note that once the buffer size is large enough to retain all the hot pages, very little savings can be made with additional increase in buffer size. Even with higher affinity, the savings are significantly lower (marked with 'U') under uniform access pattern within each DB cluster. Under skewed access pattern, even for lower transaction affinity the hot pages are retained across all buffers, and hence, a significant write I/O savings can be made under the DF policy with the increase in buffer size. Here, the updated hot pages move from node to node. However, under the DTF policy, a larger fraction of remote accesses due to lower transaction affinity causes a larger fraction of writes to be propagated to the disk, and hence, very little savings can be made.

Comparison of the CA, SRS and DTF Policies: Here, we compare the performance of the CA, SRS and DTF policies only. Since, the relative performance of DTF and DF policies are well understood, we drop the DF policy in this comparison list. The CA policy retains no form of strong lock, while the SRS policy retains only the S locks and the DTF policy retains both S and X locks. Clearly, the update probability has a strong impact on their relative performances, as it determines the fractions of locks retained under the SRS policy which is halfway between the CA and the DTF policies. Figure 5 shows the effect of update probability on the response times of these three policies, for higher and lower transaction affinity (0.9, 0.4) cases. For high transaction affinity, the response times of all three policies are quite close, and the change in update probability has a smaller effect on their response times. This is due to the fact that 1) the local buffer hit probability changes by a small amount as a result of a lower buffer invalidation rate, 2) the remote buffer hit probability is also small due to a higher transaction affinity, and 3) most of the write propagation is asynchronous (except a small synchronous propagation delay under the DTF policy). However, the above effects become very significant for lower transaction affinity (see dashed curves). Under the DTF policy, as more dirty pages referenced are retained in the remote buffers, its response time increases due to write I/O propagation delay. The CA policy suffers from lower local buffer hit probability due to a lower transaction affinity. However, the update probability has a relatively smaller impact on its response time. Note that the SRS policy maintains its lowest response time in all cases. The SRS policy takes advantage of remote buffer hit and also eliminates some global lock requests through S lock retention. Unlike the DTF policy, it does not defer any writes and hence, does not suffer from any synchronous write I/O propagation delay (similar to the DF policy). The SRS policy should be preferred not only for its performance advantage but also for its simpler recovery characteristic.

Figure 6 shows the resultant trade-offs in response time and maximum transaction throughput for a higher and lower affinity cases. Both the SRS and DTF policies provide lower response times for lower transaction rates or CPU utilizations by incurring remote messages and hence, lower maxi-

mum throughputs for the lower affinity case. Note that for a lower transaction affinity, the CPU utilization of the CA policy is lower than that of the other two policies as it incurs no remote message overhead. However, the CA policy suffers from low buffer hit probability and provides very high response time. For the high affinity case, both the SRS and DTF policies perform better in terms of both response time and maximum throughput.

Comparison of the CA, DF and ODF Policies: The DF policy can provide both lower response time and higher maximum throughput (except for lower affinity) by accessing remote buffers without forcing the writes. However, it still does not provide the best buffer utilization, as it allows the shared pages to be replicated across multiple nodes. The ODF policy (through one copy constraint) can improve the buffer utilization at the expense of more remote accesses. This improves the response time for lower CPU utilizations, but reduces the maximum throughput in comparison to the other two policies (not shown due to space limitation). For high affinity, the replication of pages across multiple nodes is rare, since most pages are accessed and retained at the affinity site. Therefore, the effect of buffer size on the response times for the CA, DF and ODF policies is shown in Figure 7 only for a lower affinity (0.4) case. The solid and the dashed curves are for a low and moderate update probabilities (0.01, 0.2). (Note that for a high probability of update the performances of DF and ODF policies become closer, as very few shared pages are replicated and retained in multiple buffers under the DF policy.) The local and overall buffer hit probabilities for the corresponding cases are shown in Figures 8 and 9. Note that both local and overall buffer hit probabilities for the ODF policy remain unchanged due to the change in update probability. This is because a page is lost from the local buffer irrespective of the remote access being read or write. The local buffer hit probability is lower compared to the other two policies as fewer hot pages are retained in any local buffer. However, the overall buffer hit probability is the highest since the most number of distinct pages are retained across all buffers (no replication). The difference in local buffer hit probabilities between the CA and DF policy is similar to that of *detection* and *notification* policies in [6]. The difference is quite small for most cases except for a uniform access pattern and moderate buffer size (see [6] for more details).

5 Summary and Conclusions

Buffer coherency control can be achieved through retaining a lock (shared, exclusive, etc.) on each page in the buffer, even after the requesting transaction has committed. Depending upon the lock mode held for retention and the compatibility of lock modes specified, different retention policies can be devised. In addition to tracking the validity of the buffered granules, additional capabilities can be provided such as deferred writes to support no-force policy on commit, (node) location identification of valid granules to support remote memory accesses, and shared/exclusive lock retention to reduce the number of global lock requests for concurrency control. However, these can have serious implications not only on the performance but also on the recovery complexity.

In this paper, five different integrated coherency policies are considered. We classify these policies into three different categories according to their recovery requirements. Analytic models are developed to study the trade-offs on both throughputs and response times of the policies with a similar level of recovery complexity and the performance gain achievable through increasing the level of recovery complexity. Various factors including buffer size, transaction rate, update probability and data access pattern are examined. We also capture the notion of affinity that is shown in this paper to have a strong impact on the performance of these policies.

For the policies with simple recovery characteristic, besides the check-on-access (CA) policy which serves as our base policy, we also consider a shared lock retention (SRS) policy under the force scheme. Two of the policies considered using deferred writes are with medium and complex recovery characteristics: Deferred until Transfer or Flushing (DTF) and Deferred until Flushing (DF) polices, respectively. Both the DF and DTF polices outperform the CA policy under high affinity. However, at low affinity the CA policy can provide higher maximum throughput. At the expense of its recovery complexity, the DF policy consistently does better than the DTF policy in both response time and maximum throughput, especially for the case with lower affinity. In contrast to the DF policy, the DTF policy is very sensitive to the update probability. What is interesting to note is that the SRS policy with simple recovery characteristics provides better performance than the DTF policy in most cases. It shows more robustness to the affinity factor and less sensitivity to the update probability as compared to the DTF policy.

We propose additional variations in policies based on a single copy constraint, i.e. no page is allowed to be buffered by more than one node. Without any loss of generality, we consider this variation, the ODF policy, only for the DF policy, i.e. the one with complex recovery characteristic. The ODF policy can provide better overall (local + remote) buffer hit probability at the expense of CPU overhead than the DF policy. That is to say the ODF policy can lead to smaller response time for lower CPU utilizations than the DF policy, especially for the case with smaller buffer size, albeit with smaller maximum throughput.

References

[1] Behman, S.B., T.A. DeNatale, and R.W. Shomler, "Limited Lock Facility in a DASD Control Unit", *Tech. Report TR 02.859*, IBM General Products Division, San Jose, CA, Oct. 1979.

[2] Bennett, B. T., P. A. Franaszek, J. T. Robinson, and P. S. Yu, "Check on Access Via Hierarchical Block Validation", *IBM Technical Disclosure Bulletin*, Vol. 27, No. 7A, Dec. 1984, pp. 3752–3756.

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

[4] Dan, A., "Performance Analysis of Data Sharing Environments", *PhD Dissertation*, University of Massachusetts, Amherst, September 1990. Also an ACM Distinguished Dissertation, 1991, (to be published by the MIT press).

[5] Dan, A., D. M. Dias, and P. S. Yu, "The Effect of Skewed Data Access on Buffer Hits and Data Contention in a Data Sharing Environment," *16th Int. Conf. on VLDB*, Brisbane, Australia, Aug. 1990, pp. 419–431.

[6] Dan, A., and P. S. Yu, "Performance Analysis of Buffer Coherency Policies in a Multi-System Data Sharing Environment," *IEEE Trans. on Parallel and Distributed Systems*, (to appear), also *IBM Research Report RC 16361*, Nov. 1990.

[7] Dan, A., and P. S. Yu, "Performance Analysis of Coherency Control Policies through Lock Retention", *IBM Research Report, RC 17663*, 1992.

[8] Dias, D. M., B. R. Iyer, J. T. Robinson, and P. S. Yu, "Integrated Concurrency-Coherency Controls for Multisystem Data Sharing," *IEEE Trans. on Software Engineering*, Vol. 15, No. 4, April 1989, pp. 437–448.

[9] Howard, J. H. and et al, "Scale and Performance in a Distributed File System," *ACM Transactions on Computer System*, Vol. 6, No. 1, Feb. 1988, pp. 51–81.

[10] Kronenberg, N., H. Levy and W. D. Strecker, "VAXcluster: a Closely-Coupled Distributed System," *ACM Trans. on Computer System*, Vol. 4, No. 2, pp. 130–146, May 1986.

[11] Li, K., and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Transactions on Computer System*, Vol. 7, Nov. 1989, pp. 321–359.

[12] Mohan, C., and I. Narang, "Recovery and Coherency Control Protocols for Fast Intersystem Page Transfer and Fine Granularity Locking in a Shared Disks Transaction Environment", *17th Int. Conf. on VLDB*, Barcelona, Spain, Sept. 1991, pp. 193–207.

[13] Mohan, C., and I. Narang, "Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment", *Proc. of Int. Conf. on Extending Data Base Technology*, Vienna, March 1992 (to appear), also *IBM Research Report RJ 8301*, Aug. 1991.

[14] Ramachandran, U., M. Ahamad and M. Y. A. Khalidi, "Coherence of Distributed Shared Memory: Unifying Synchronization and Data Transfer", 18^{th} *Int. Conf. on Parallel Processing*, St. Charles, Ill, Aug. 1989, pp. II-160–II-169.

[15] Rahm, E., "Primary Copy Synchronization for DB-Sharing," *Information Systems*, Vol. 11, No. 4, 1986, pp. 275–286.

[16] Rahm, E., "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Data Sharing," *IBM Research Report, RC 14325*, 1988.

[17] Robinson, J.T., "A Fast General-Purpose Hardware Synchronization Mechanism", *SIGMOD Record*, 1985, pp. 122–130.

[18] Strickland, J. P., P. P. Uhrowczik and V. L. Watts, "IMS/VS: An Evolving System," *IBM Systems Journal*, Vol. 21, No. 4, pp. 490–510, 1982.

[19] Teng, J., and R. Gumaer, "Managing IBM Database 2 Buffers to Maximize Performance", *IBM Systems Journal*, Vol. 24, No. 2, 1984.

[20] Wilkinson, K., and M. A. Neimat, "Maintaining Consistency of Client-Cached Data" *16th VLDB Conf.*, Brisbane, Australia, August 1990, pp. 122–133.

[21] Yu, P. S., Dias, D. M., Robinson, J. T., Iyer, B. R. and Cornell, D. W., "On Coupling Multi-Systems Through Data Sharing", *Proceedings of the IEEE*, Vol. 75, No. 5, May 1987, pp. 573–587.

[22] Yu, P. S., and A. Dan, "Impact of Affinity on the Performance of Coupling Architectures for Transaction Processing", *IBM Research Report 16431*, Jan. 1991.

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

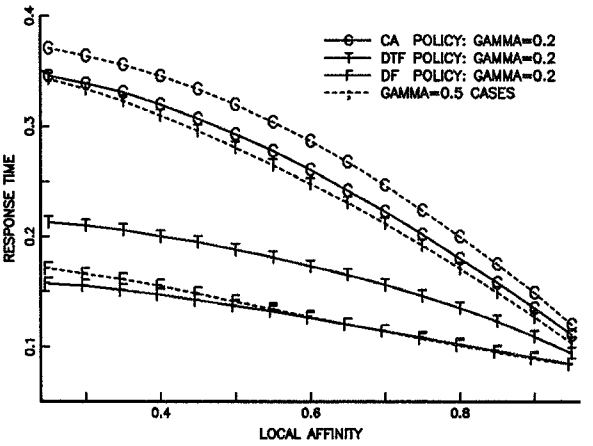

Figure 1: Effect of affinity (access rule:80-20, λ = 40 tps/node)

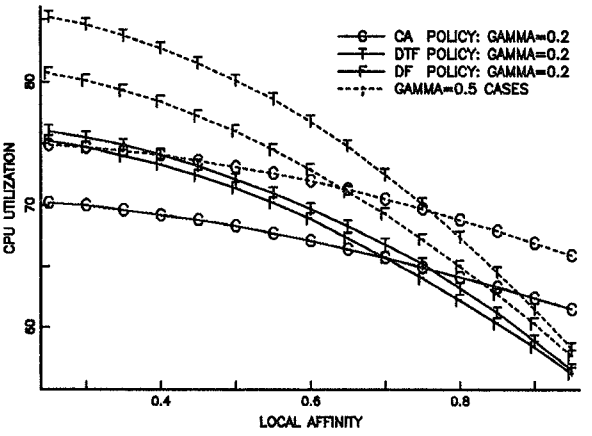

Figure 2: Effect of affinity (access rule:80-20, λ = 40 tps/node)

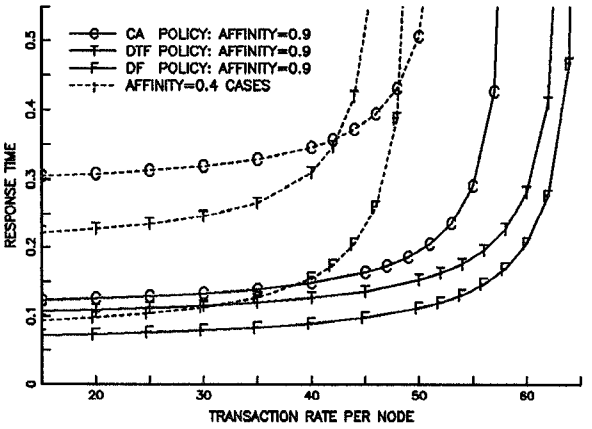

Figure 3: Comparison of policies (γ = 0.5, access rule:80-20)

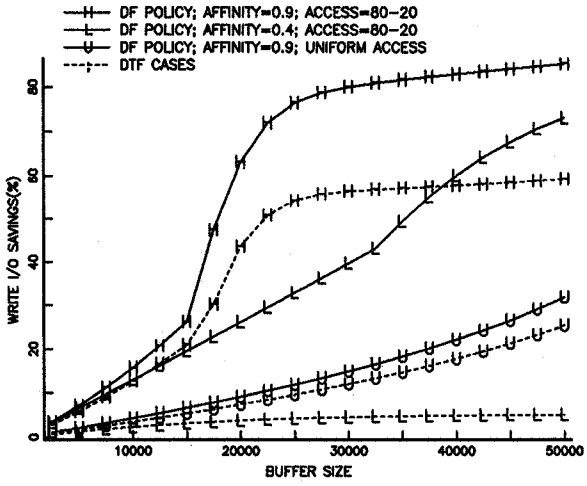

Figure 4: Comparison of write I/O savings ($\gamma = 0.2$)

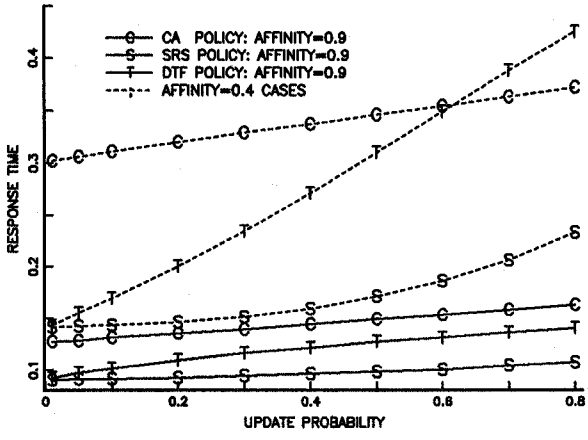

Figure 5: Effect of update probability, γ (access rule:80-20, $\lambda = 40$ tps/node)

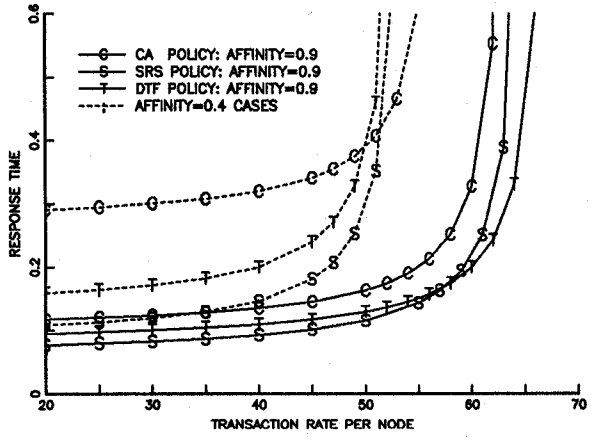

Figure 6: Comparison of policies ($\gamma = 0.2$, access rule:80-20)

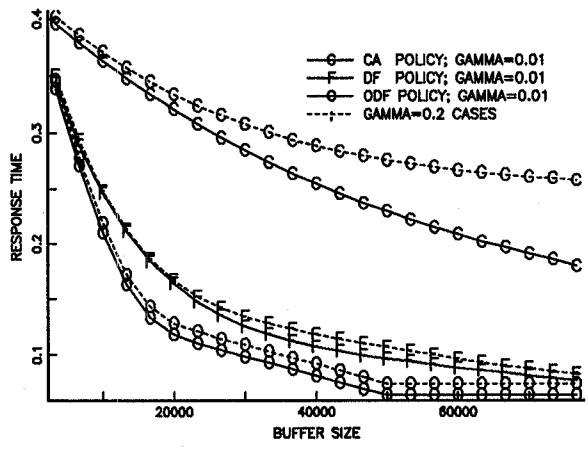

Figure 7: Effect of buffer size on response time (access rule:80-20, affinity=0.4, $\lambda = 40$ tps/node)

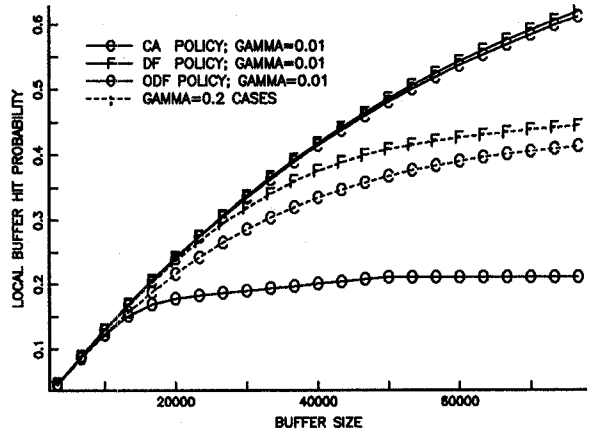

Figure 8: Comparison of local buffer hit probabilites (access rule:80-20, affinity=0.4)

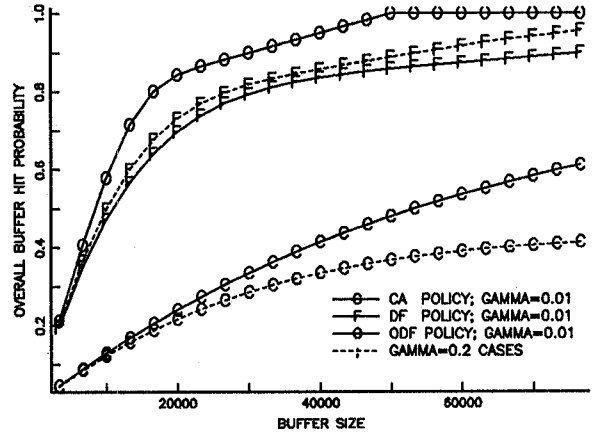

Figure 9: Comparison of overall buffer hit probabilites (access rule:80-20, affinity=0.4)