Performance Analysis of Buffer Coherency Policies in a Multisystem Data Sharing Environment

Asit Dan, *Member, IEEE,* and Philip S. Yu, *Fellow, IEEE*

Abstract— **In this paper, we compare six buffer coherency policies for a multisystem transaction processing environment. These policies differ in their basic approaches on how and when the invalidated pages are identified or if the updated pages are propagated to the buffers of the remote nodes. They can be classified as *detection*, *notification* (of invalid pages), and (update) *propagation* oriented approaches. The policies trade off CPU overhead of coherency messages with buffer hit probability in different ways, resulting in a tradeoff of response time and maximum throughput. The main contribution of this paper is to develop analytical models to predict buffer hit probabilities under various buffer coherency policies assuming the LRU buffer replacement policy and the Independent Reference Model (IRM). The buffer models are validated using simulation models and show excellent agreement. We also develop integrated analytic models capturing buffer hit probability and CPU overhead to predict the overall response times under these coherency policies. We find the difference in buffer hit probabilities amongst various policies to be very sensitive to the skewness of the data access. Under nonuniform access pattern the detection oriented policies that incur the smallest overheads are preferred.**

Index Terms—**Analytic model, buffer coherency, data sharing, performance analysis, transaction processing.**

I. Introduction

THE coherency problem of software managed buffers is common to several multisystem 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 IO operations or remote data accesses. Examples are transaction processing environments where multiple nodes share a common database [33], [25], [30], [39], [10], [28], client-server environments where client nodes may access the same set of files stored in the server [22], [37], [7], and the distributed shared memory environments where physical pages corresponding to a virtual page may be present at multiple computing nodes [26], [29], [5], [4]. The coherency policy ensures that the data pages present in multiple local buffers are up to date, and hence, if a data page is updated by any node either its copy is propagated to all other nodes or the old copies of this page present in remote local buffers are invalidated. Different buffer coherency policies can be devised to address this issue.

In this paper, we focus on developing analytic models to predict buffer hit probability as well as overall response time under various coherency policies, assuming a transaction processing application in a multisystem data sharing environment. This approach is general and the model can be used to analyze other similar environments like a client-server environment or a distributed shared memory environment, albeit the parameter values, operation constraints and (both hardware and software) implementations may be quite different from that of the transaction processing environment. The analytical models are used to analyze the tradeoff of coherency overhead and local buffer hit probability under different coherency control policies.

We compare six buffer coherency policies which differ in their basic approaches on how and when the invalidated pages are identified or if the updated pages are propagated to the buffers of the remote nodes.[1] The policies can be classified broadly into three categories: *detection* of invalidated pages, *notification* of invalidated pages, and *propagation* of updated pages. Under the *notification* oriented approach, messages consisting of the identities of the updated pages are sent to the remote nodes by the updating node at the transaction commit time so that the remote nodes can invalidate the obsolete copies of these pages if present in their local buffers. In contrast, the *detection* oriented approach requires no such messages to be sent to the remote nodes but requires the validity of a page to be checked by the accessing node at the page reference time. Under the *propagation* oriented approach, the updated pages are propagated to the remote nodes at the transaction commit time. Both the second and third approaches (*notification* and *propagation*) can have variations based on whether the messages are broadcast to all other nodes or sent (selectively) only to the nodes with a copy of the invalidated pages in their buffers. The selective notification or selective propagation requires that knowledge of the copies of a data page present in multiple buffers be maintained by a centralized coherency manager. We also propose a mixture of the first two (detection and notification) approaches, where the coherency is maintained through detection while a delayed batched notification is used to improve the buffer hit probability. Based on these three approaches and their combinations, six policies are considered: check-on-access, broadcast invalidation, selective notification of invalidation,

Manuscript received April 8, 1991; revised October 28, 1991.

The authors are with the IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598.

IEEE Log Number 9205851.

[1] All the policies that are considered here are equally applicable in both transaction processing and nontransaction processing environments. However, in a transaction processing application, how and when the updated pages are propagated to the disks has implications for recovery complexity. In this paper, we assume that the updated pages are propagated to the disk at the transaction commit time. Other variations of coherency policies that consider deferred writes and retention of locks beyond transaction commit can be found in [30], [31], [28], and [15].

broadcast update, selective notification of update, and check-on-access with periodic notification policies.

The various policies trade off CPU overhead of coherency control messages with buffer hit probability in different ways resulting in a tradeoff of response time and maximum throughput. The update propagation policies incur the highest CPU overhead, followed by the notification oriented approach. In contrast, the detection oriented approach tends to have the lowest buffer hit probability, since undetected obsolete pages may continue to stay in the buffer. The notification oriented approach can make better use of the buffer space at the nodes, since invalidated pages are immediately purged. The buffer hit probability of the update propagation approach does not suffer from invalidation effect, since the newly updated copy merely replaces the old copy of that page. Therefore, the three approaches can be ranked in terms of increasing buffer hit probabilities and increasing coherency overhead as i) the *detection* approach, ii) the *notification* approach, and iii) the *propagation* approach. However, the absolute values of the differences on throughput and response time depend on the CPU overhead of sending and receiving a message, and the buffer hit probability, which in turn depends on the buffer size, the access pattern (uniform versus nonuniform), the buffer replacement policy, the number of nodes, and the probability of update. Therefore, we develop analytic models to predict buffer hit probability as well as overall response time under these coherency policies, assuming a transaction processing application and the LRU buffer replacement policy. The buffer models are validated through detailed simulations and the analysis results show excellent agreement with the simulations results. We also show the generality of these models by considering both homogeneous and nonhomogeneous workload models, i.e., where the database access pattern may or may not be the same for all nodes. We find the difference in buffer hit probabilities to be very sensitive to the skewness in the data access, and further affected by the number of nodes, update probability, and the buffer size. Furthermore, under nonuniform access pattern the difference in buffer hit probabilities between the detection and notification approaches becomes small, and the detection oriented approach is preferred. Our proposed policy takes advantage of both the detection and notification approaches, and provides both the lower response time and higher maximum throughput.

The paper is organized as follows. Section II presents some earlier work both in terms of proposed policies as well as their analyses. Section III describes the multisystem data sharing environment and the details of the the six buffer coherency policies that are considered in this paper. The buffer model, and its integration with the concurrency control and system resource models, are developed in Section IV. Model validation, and projections from the model are presented in Section V. A summary and concluding remarks appear in Section VI.

II. Earlier Work

There are few existing analytical models for the database buffer particularly for a multisystem environment. Previous models of the database buffer for multisystem data sharing [39], [17] have been empirical, based on trace driven simulations. In [23] performance of two of the policies (broadcast invalidation and check-on-access) that we consider here, are studied under a uniform access workload using a simulation model. The importance of considering skewed data access for database applications is discussed in [8] and [14]. We explicitly model an LRU buffer capturing skewed access, and cross-invalidation to predict the buffer hit probability. We extend the models developed in [13] for the analysis of broadcast invalidation policy, which is a notification oriented approach, to analyze other coherency policies. The analysis of buffer hit probability under the detection oriented approach is more complex due to the presence of undetected invalid pages scattered in the LRU buffer. A decomposition approach is taken to model overall system performance, where the buffer model, the two-phase locking concurrency control (CC) model and the hardware resource model are first developed separately and then solved simultaneously using an iterative procedure. The approach has been extensively validated in [10], [11], and [13]. The CC and system resource models used are extensions for skewed data access of the models in [42], [41], and [40] where the decomposition approach is first introduced.

We mention in passing that there are a number of studies in the literature on multiprocessor cache coherency based on either snooping-cache architecture [2], [21], [36], [38], [20] or directory based architecture [1]. In general, either the workloads (database access skew versus temporal locality of a process) are different for the database buffer and multiprocessor cache environments, or the cache replacement policies considered in some of these works are simpler (random replacement or direct mapping). An analytical model for cache hit degradation due to multiprocessor invalidation effect is developed in [19] under the LRU replacement policy where values of cache hit for various cache sizes without the invalidation effect is assumed to be known *a priori*. Apart from the workload models and buffer models, the system parameters (processor cycle needed for invalidation versus message overhead) are also very different. The closest environments to the transaction processing environment that we consider here are the client-server environment and the distributed shared memory environment. A number of protocols similar to some of the protocols studied here are proposed for these environments. Two of the notable schemes are the selective notification scheme [26] and selective propagation of updates [29] (both are detailed in the next section). The Munin system uses different coherency mechanism for different data types to optimize performance [5].

In [37], a simulation study on a client-server environment has been reported to compare two cache coherency algorithms which are integrated with the lock manager. These two algorithms, referred to as cache locking and notification locking, are in similar concept to the detection and propagation oriented policies considered here. However, in contrast to the buffer model in this paper that captures skewed data access pattern, the above study does not model an LRU buffer but, instead, fixes the contents of the client's buffer and assumes a uniform access pattern. Also, due to the nature of the

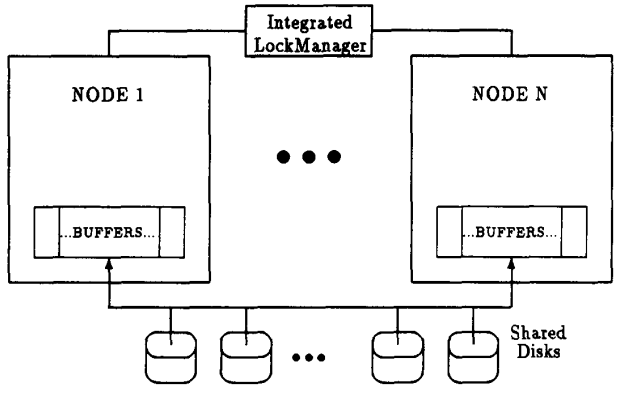


Fig. 1. Multisystem data sharing environment.

distributed environment the concurrency control employed is semi-optimistic instead of two-phase locking considered in this paper. An independent recent work by Carey *et al.* [7] studies an environment closer to the environment in [37], and is also based on simulation.

Finally, we mention in passing that buffer coherency can also be maintained if all pages used are purged from the buffer at transaction commit time. This is referred to as the buffer purge policy. Pages are always read from disk and there are no buffer hits. No notification or detection of page obsolescence would be needed in this case. In [6], a buffer purge policy is examined in a simulation study. The environment considered there is with a very large database size relative to the buffer size, and hence, the achievable buffer hit probability under any policies would be quite small. Otherwise, the policy would do poorly compared with other policies as in the environment we consider here. We are not going to examine the buffer purge policy in this paper.

III. The Multisystem Data Sharing Environment

The multisystem data sharing environment considered consists of multiple loosely coupled nodes sharing a common database at the disk level (Fig. 1). The processing of a transaction is modeled as consisting of execution and commit phases. Standard two-phase locking is used for concurrency control where a centralized lock manager is assumed to be available which can be either implemented on control units [3] or through some specialized processor [32]. Each node maintains a local buffer and caches a part of the database in this buffer to reduce the transaction response time. An LRU buffer replacement scheme is used by each node for its local buffer management. To access a data page, a transaction requests a copy of the item from the local buffer manager. The buffer manager returns a copy of the page to the requesting transaction if the page is present in the buffer. Otherwise, a copy of that page is brought in from the shared disk to the local buffer. In either case, the newly accessed page is placed at the top of the LRU stack. In the case when a new page is brought in from the disk, if there is no free buffer available, then the page at the bottom of the LRU stack is pushed out of the stack. During the execution of a transaction, its updates are made on its local copies. The updated copies are made permanent replacing the old copies, both in the local buffer and the shared disk, at the commit time of a transaction.

A. Buffer Coherency Policies

The copies of the same page may be present at more than one node. Therefore, a coherency control mechanism is needed to prevent obsolete pages from being accessed. As mentioned earlier, there are three fundamentally different approaches to solve the coherency problem: *detection* of invalidated pages, *notification* of invalidated pages, and *propagation* of updated pages. Based on these three approaches and their combinations, we consider six different coherency control policies. The implementations of these policies (i.e., additional data structures maintained, etc.) are detailed below. These policies differ in their tradeoffs between the CPU overhead of sending, receiving and processing of coherency messages and the buffer hit probability. Generally speaking, immediate notification on page obsolescence or propagation of updated pages causes CPU overhead, and later detection reduces buffer hit probability.

1) Detection Approach:

Check-on-Access (CA) Policy: Under this approach, the obsolete pages are detected at the page access time by a transaction. We assume an integrated Concurrency Coherency (CC) manager/controller [18] which not only provides the traditional concurrency control service, but also tracks which node has a valid copy of a page. The lock table entry can contain an additional *valid bit* for each node to indicate whether it has a valid copy. Before accessing any page, the processing node of the transaction makes a global lock request to the integrated CC manager. In response, the integrated CC manager returns not only the requested lock, but also the result of the associated buffer validation check based on the valid bit. Note that if the valid bit is off, it will be turned on after the status of the valid bit is returned as invalidation has now been accomplished. At the lock release time, if the page has been updated, the valid bits of all other nodes 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 early notification of invalidated pages, but it also reduces the buffer hit probability as the obsolete pages continue to reside in the local buffer. An additional advantage of the CA policy is that locks are released sooner at transaction commit time as there are no invalidation messages or acknowledgment to be sent or received.

2) Notification Approach:

Broadcast Invalidation (BI) Policy: Under this approach, an invalidation message containing a list of pages modified by the committing transaction is broadcast to all other nodes during transaction commit [33], [13]. Each node upon receiving the invalidation message will check for the old copies of the updated pages and mark them invalid if present in its local buffer. This immediately frees up the buffer space occupied by the obsolete pages and page frames are brought to the bottom of the LRU stack. The locks on updated pages will be held until the acknowledgment to the invalidation messages are received from all other nodes. However, we assume that the transaction response can be sent to the terminal user and the transaction can be committed after the updates are reflected to disks, and before the invalidation message is broadcast. The overhead for notification, check for invalidation and acknowledgment under this policy can become excessive as the number of nodes increases.

Selective Notification of Invalidation (SNI) Policy: The number of invalidation messages as well as processing overhead of unnecessary messages can be reduced by selectively notifying only the nodes that holds at least one old copy of the updated pages. This requires that the updating node has the information about which other nodes hold one or more old copies of the updated pages. The policy can be implemented under an integrated CC manager as mentioned for the CA policy. The list of nodes having a valid copy of a page can be obtained from the integrated CC manager (based on the valid bits) at the time when an exclusive lock on the page is granted by the updating node. At transaction commit time, based on the pages updated by a transaction, the processing node can determine the list of nodes that need to be notified for page invalidation. A message is sent to each of these nodes with a list of the specific pages to be invalidated. Similar to the BI policy, the locks on updated pages will be held until acknowledgment to all invalidation messages are received. Finally, it is assumed that the addresses or identities of the pages flushed out from the local LRU stack can be piggybacked with the next lock request to the integrated CC manager with no additional overhead. The integrated CC manager can then turn off the corresponding valid bit for the node in the lock table entry to indicate that it no longer has a valid copy. Note that this is not required for the CA policy as the integrated CC manager only indicates that if the node has a copy then it would be valid. For the SNI policy, if this is not done, invalidation messages will be sent to a node long after the pages have been purged from its buffer and the message overhead would increase.

3) Propagation Approach:

Under this approach, at transaction commit time, the updating node broadcasts not only the identities of the updated pages, but also a copy of the updated pages to all other nodes. Those nodes currently owning an obsolete copy of an updated page will replace it with the fresh copy. Otherwise, the pages propagated will be ignored. Like the notification approach, depending upon whether or not the integrated CC manager is available, two different policies are possible, namely the *Broadcast Update (BU) policy* and the *Selective Notification of Update (SNU) policy,* where updates are propagated to all or selected set of nodes, respectively.

4) Hybrid Approach:

Check-on-Access with Periodic Notification (CAPN) Policy: The last policy considered, CAPN, uses a combination of the detection approach used by the CA policy and the notification approach used by the SNI policy. The check on access part of the policy is the same as in the CA policy. Each time a page is accessed, a buffer validity check (i.e., the CA request) is combined with the lock request made to the integrated CC manager. The integrated CC manager now maintains an additional *invalid page list* for each node of the pages to be invalidated in its local buffer. At the lock release time, if the page has been updated, the valid bits corresponding to all other nodes associated with this lock table entry will be turned off. Each time a valid bit is turned off in a lock table entry, the address or identity of the page is also added to the invalid page list of the corresponding node. It is assumed that this list can be piggybacked with the response to the next lock request from the corresponding node. The invalid page list of that node would then be reset to null at the integrated CC manager. Upon receiving the list, the local buffer manager can then purge the invalidated pages from its buffer. Compared with the SNI or BI policy, the invalidation notification is delayed, but the check on access part of the policy maintains the integrity of the data access. There will be no race condition as long as the requests from each node are processed and returned in order. Note that if the invalid page list is not piggybacked with the reply to the next lock request, then the integrated CC manager must remove the corresponding entry (if present) from the invalid page list of the requesting node, since the invalidation is now performed through the CA request. Otherwise, a local buffer manager may first read in a fresh copy replacing the invalid copy after the check on access is done, and then invalidate the fresh copy again when the list of invalidated pages finally arrives.

IV. Performance Model

We first develop the buffer model under each of the policies considered in Section III and will then briefly describe the overall system model. The data sharing system considered consists of N loosely coupled nodes. We assume that the database consists of D pages. Each node maintains a local buffer of size B. Transactions arrive at each node according to a Poisson process with rate λ. We assume that each transaction accesses L pages from the shared database. Each page access is assumed to be independent of all other page accesses [Independent Reference Model (IRM)]. Such a model holds for many database transaction processing applications where each transaction accesses a relatively small set of pages. For instance, in the TPC-A workload, each transaction performs a

debit/credit operation to an account chosen independently and equiprobably from the account relation [35]. Further support for the random access pattern in a general database transaction processing workload can be found in [24] and [14]. Though independent, the database accesses need not be uniform over all pages. In the above TPC-A example, index pages to the accounts are accessed more frequently than account pages.

A. Buffer Model

We will consider two different skewed access workload models characterized by the access pattern at different nodes. The first is the case of a homogeneous data sharing system where the access pattern to the database is the same from all nodes. To show the generality of our modeling methodology, we also consider a second workload model where each node has its own favorite data set which is accessed less often by others.

1) Homogeneous Node Access Pattern: The skewed access pattern is modeled in earlier work as access to two kinds of data [34] (*hot* data and *cold* data). A common model is the so-called 80-20 rule, where 80% of the accesses go to a hotset that comprises 20% of the database, as illustrated in Fig. 2. As shown in the figure, the frequency of access to each page is much higher in the hotset (16 times in this example) than that of pages in the coldset, but is uniform within each set. For analysis purposes we refer to this as a two partition model since the data can be logically grouped into two partitions based on their distinct access frequencies. In general, the access pattern can be approximated as consisting of multiple such partitions. Based on the frequency of data access, the data pages are grouped into P partitions, such that the probability of accessing any page within a partition is uniform. Let β_i denote the fraction of the database in partition i, i.e., the size of partition i is $\beta_i D$. For example in Fig. 2, there are two partitions and β_1 and β_2 are given by 0.2 and 0.8, respectively. Let α_i denote the probability that a database access is to a page in partition i, such that, $\sum_{i=1}^{P} \alpha_i = 1$. Hence, the relative access frequency for the pages in partition i is $\alpha_i/(\beta_i D)$. In our example, α_1 and α_2 are given by 0.8 and 0.2, respectively. The probability that a page is updated is allowed to depend on the partition to which it belongs. The probability that a page accessed from the ith partition is also updated is denoted as p_{u_i}. Thus, the average rate at which pages of partition i are updated is given by $N\lambda L\alpha_i p_{u_i}$. We assume that there is no shared buffer in the system. Table I provides a summary of the symbols used in this paper.

Since the data sharing system is assumed to be homogeneous, we focus our attention on a single buffer. We first look at the buffer hit probability under the detection oriented approach (CA) which is more difficult to analyze as both valid and invalid pages are mixed together in the buffer. Our analysis extends the methodology used in [13] for estimating the buffer hit probability under the BI policy. To estimate the steady-state probability of buffer hit, we first derive the average number of valid pages of each partition in the local buffer of any node. Let $V_i(j)$ denote the average number of valid pages of partition i in the top j locations of the LRU stack. Therefore, the buffer hit probability of the ith partition is $h_i = V_i(B)/(\beta_i D)$, and

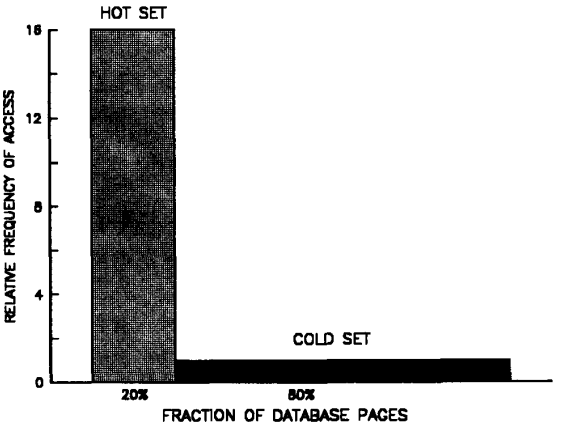


Fig. 2. Illustrations of the access frequency under 80-20 rule.

the overall buffer hit probability for a page requested by a transaction is estimated as

$$H_{ca} = \sum_{i=1}^{P} \alpha_i h_i. \tag{1}$$

Let $p_i(j)$ be the probability that the jth buffer location from the top of the LRU stack contains a page of partition i. Let $p_i^v(j)$ be the probability that the page is valid given that the page belongs to the ith partition. Then,

$$V_i(j) = \sum_{l=1}^{j} p_i(l) p_i^v(l). \tag{2}$$

Also, let $Y_i(j)$ denote the average number of (both valid and invalid) pages of partition i in the top j locations of the LRU stack. Then,

$$Y_i(j) = \sum_{l=1}^{j} p_i(l). \tag{3}$$

We will set up a recursive formulation to determine $p_i(j+1)$ and $p_i^v(j+1)$ for $j \geq 1$ given $p_i(l)$ and $p_i^v(l)$ for $l = 1, \cdots, j$. Consider a smaller buffer consisting of the top j locations only. The buffer location $(j+1)$ receives the page that is pushed down from location j. Let $r_i(j)$ be the rate at which pages of partition i are pushed down from location j. Similarly, let $r_i^v(j)$ be the rate at which valid pages of partition i are pushed down from location j. Our estimation of $p_i(j)$ and $p_i^v(j)$ are based on the following two observations.

Conservation of flow: Under steady-state conditions, the long term rate at which pages of the ith partition get pushed down from the top j locations of the buffer equals the rate at which they are brought into the top j locations. Note that if the new access is a hit on an invalid page present in the top j buffer locations, the page simply will be refreshed and will not cause a replacement from location j. Hence, the push down rate, $r_i(j)$ is given by

$$r_i(j) = \lambda L \alpha_i \left[1 - \frac{Y_i(j)}{\beta_i D}\right]. \tag{4}$$

TABLE I
SUMMAY OF THE SYMBOLS USED

System Environment:	
N	*Number of nodes*
B	*Buffer size per node*
M_{cpu}	*Processor speed in MIPS*
T_{io}	*I/O access delay*
T_{log}	*Log I/O delay*
Workload:	
D	*Number of database pages*
P	*Number of database partitions*
α_i	*Prob. of accessing i^{th} partition*
β_i	*Fraction of database lies in partition i*
p_{u_i}	*Probability of update for pages of partition i*
λ	*Transaction arrival rate per node*
L	*Number of pages accessed per transaction*
N_{setup}	*Number of I/O for initial loading of trx.*
I_{trx}	*Base Trx. pathlength*
I_{io}	*Instruction overhead per I/O operation*
I_{commit}	*Instruction overhead for trx. commit*
I_{msg}	*Overhead per message processing operation (SEND or RECEIVE)*
I_{lmsg}	*Extra Overhead per long message operation*
I_{search}	*Overhead per hash table look up*
I_{purge}	*Overhead per hash table purge*
I_{copy}	*Overhead per page copy*
Other symbols:	
$p_i(j)$	*Prob. that buffer location j holds a page of partition i*
$p_i^v(j)$	*Prob. that buffer location j is valid given that it holds a page of partition i*
$Y_i(j)$	*Number of pages of partition i in the top j buffer locations*
$V_i(j)$	*Number of valid pages of partition i in the top j buffer locations*
$r_i(j)$	*Push down rate for all pages of partition i from buffer location j*
$r_i^v(j)$	*Push down rate for valid pages of partition i from buffer location j*
h_i	*Buffer hit probability of i^{th} partition*
H_X	*Overall buffer hit probability under policy X*
$p_{brdcast}$	*Probability that a trx. requires a broadcast*
N_{upd}	*Average number of pages updated per trx.*
N_{purge}	*Averages number of pages invalidated in any remote buffer per trx.*
N_{msg}	*Average number of invalidation messages sent per trx. under SNI policy*
I_{cpu}	*Total CPU demand (execution+overhead) over all nodes per trx.*
ρ	*CPU utilization*
P_{cont}^i	*Prob. of contention for lock on a page of i^{th} partition*
R_w	*Mean waiting time per lock contention*
R_{exec}	*Mean trx. execution time*
R_{Commit}	*Mean trx. Commit time*
R_{inval}	*Mean time to send out invalidation message and receive all acknowledgements*
R	*Mean trx. response time*

Using a similar conservation of flow argument for the valid pages, we equate the long term rate at which valid pages of the ith partition get pushed down from the top j locations of the buffer, $r_i^v(j)$, to the difference of the rate at which they are brought into the top j locations, and the rate they become invalid. Hence, $r_i^v(j)$ is given by

$$r_i^v(j) = \lambda L \alpha_i \left[1 - \frac{V_i(j)}{\beta_i D}\right] - (N-1)\lambda L \alpha_i p_{u_i} \frac{V_i(j)}{\beta_i D}. \quad (5)$$

Relative push down rate: The expected value of finding a page of the ith partition in the $(j+1)$st buffer location over all time, $p_i(j+1)$, is approximately the same as the probability of finding a page of the ith partition in the $(j+1)$st buffer location in the event that a page is pushed down from location j to location $(j+1)$. Formally, *Prob.*$\{$*location* $(j+1)$ *contains a page of partition* $i \mid$ *a page is pushed down from location* j *to location* $(j+1)\} \approx$ *Prob.*$\{$*location* $(j+1)$ *contains a page of partition* $i\}$. Hence,

$$p_i(j+1) \approx \frac{r_i(j)}{\sum_{l=1}^{P} r_l(j)}, \quad j = 1 \cdots B-1. \quad (6)$$

Note that instantaneous value of $r_i(j)$ is dependent on the content of the top j buffer locations, and the more accurate estimation of $p_i(j)$ requires the precise distribution of the content of top j buffer locations.

Similarly,

$$p_i^v(j+1) \approx \frac{r_i^v(j)}{r_i(j)}, \quad j = 1 \cdots B-1. \quad (7)$$

Equations (2), (3), (4), (5), (6), and (7) can be solved iteratively, with the base condition of $p_i(1) = \alpha_i$ and $p_i^v(1) = 1$. At the point, when $Y_i(j)$ is very close to its limit $(\beta_i D)$,

$Y_i(j)$ may exceed $\beta_i D$ because of the approximation in the above equations. This is corrected by resetting $Y_i(j)$ to $\beta_i D$ whenever $Y_i(j)$ exceeds $\beta_i D$ and $r_i(j)$ is taken to be zero for all subsequent steps for that partition. Note that, although $r_i(j)$ is a function of the transaction rate (λ), $p_i(j)$, $p_i^v(j)$ and therefore, h_i and H_{ca} are independent of λ, because λ cancels out in (6) and (7).

The buffer hit probabilities under the BI and the SNI policies are exactly the same, although the overheads incurred are different. The buffer hit probability under the CAPN policy is also roughly equal to that under the BI or SNI policy as long as the buffer size is substantially larger than the number of pages that can get invalidated between successive clean-ups of invalid pages. Let H_{bi}, H_{sni}, and H_{capn} be the buffer hit probabilities under BI, SNI, and CAPN, respectively. H_{bi} and H_{sni} can be estimated in a similar way as H_{ca} by setting the push down rate, $r_i(j)$, in (4) as the difference of miss rate and invalidation rate (buffer purge rate).

$$r_i(j) = \lambda L \alpha_i \left[1 - \frac{Y_i(j)}{\beta_i D}\right] - (N-1)\lambda L \alpha_i p_{u_i} \frac{Y_i(j)}{\beta_i D}. \quad (8)$$

(Note that there are no invalid pages under BI and SNI.) Together with (6) and (3), it can be solved iteratively with the initial condition of $p_i(1) = \alpha_i$. Hence,

$$H_{bi} = H_{sni} = \sum_{i=1}^{P} \alpha_i Y_i(B)/(\beta_i D). \quad (9)$$

$$H_{capn} \approx H_{bi}. \quad (10)$$

The buffer hit probability under the update propagation approach (BU and SNU) can be derived from either of the above two models by setting the invalidation rate to zero (reduces to the model in [12]), since the new updated pages simply replaces the old copies of them. Therefore, from (8),

$$r_i(j) = \lambda L \alpha_i \left[1 - \frac{Y_i(j)}{\beta_i D}\right]. \quad (11)$$

Together with (6) and (3), it can be solved iteratively with the initial condition of $p_i(1) = \alpha_i$.

$$H_{bu} = H_{snu} = \sum_{i=1}^{P} \alpha_i Y_i(B)/(\beta_i D). \quad (12)$$

2) Disjoint Hot Sets: The above analysis can be easily modified to capture other types of workloads. One particular workload we study here is the HOT-COLD workload of [7] where each node has its own hot set which is the cold set from the point of view of the other nodes. Let the access pattern in each node be denoted by $x - y$ (e.g., 80-10) where x fraction of accesses goes to y fraction of the database. Therefore, yD is the size of the hot set in each node and $(1-y)D$ is the size of its cold set. (That is to say the database can be considered to consist of $1/y$ partitions where each partition is the hot set of one of the nodes.) The cold set in each node consists of the hot sets of the other nodes, i.e., $(1-y)D = (N-1)yD$. Therefore, the amount of remote updates lies in the hot set of a particular node is given by $(N-1) \times \lambda L(1-x)p_u/(N-1)$ or $\lambda L(1-x)p_u$ where p_u is the probability of update for both hot and cold data. Under the notification oriented approach, the (8) can now be rewritten as

$$r_{hot}(j) = \lambda L x \left[1 - \frac{Y_{hot}(j)}{yD}\right] - \lambda L(1-x)p_u \frac{Y_{hot}(j)}{yD}.$$

$$r_{cold}(j) = \lambda L(1-x)\left[1 - \frac{Y_{cold}(j)}{(1-y)D}\right] - (N-1)\lambda L p_u \cdot \left[x + \frac{(N-2)(1-x)}{(N-1)}\right] \frac{Y_{cold}(j)}{(1-y)D}. \quad (13)$$

Note that each fraction of the cold set is a hot set in one remote node and cold set in other $(N-2)$ remote nodes. Other coherency approaches can be similarly analyzed.

B. Integrated System Model

The execution time of a transaction depends on three main factors: 1) the private buffer hit probability that determines the number of I/O operations to be performed by the transaction, 2) the concurrency control protocol used for resolving conflict in accessing data pages (waiting, abort, etc.), and 3) the processing time and the queueing delay in accessing system hardware resources such as CPU, etc. We model the private buffer hit probability, concurrency control and system resource access times separately and capture their interactions via a higher level model. This higher level model relates quantities from the lower level models through a set of nonlinear equations. The solution of the higher level model corresponds to the solution of a fixed point problem which we solve through an iterative process. The transaction execution time depends on the buffer hit probability estimated by the buffer model, and by the queueing and services estimated by the CPU model. The CC model estimates the transaction abort probability based on the transaction execution time, and this in turn affects both the buffer and resource models. (The decomposition methodology can also be similarly applied to the environment using optimistic concurrency control protocol [10], [42], [41].) We will assume the I/O subsystem is well tuned, so that mean access time can be used.

The buffer models for the various policies are given in the previous subsection. The details of the concurrency control model for locking and the resource model can be found in [39] and [41]. The transaction response time can be broken down into two parts: 1) the transaction execution phase to execute the transaction application and access the databases with its average time denoted as R_{exec}, and 2) the commit phase to write the updates and log onto disk with its average time denoted as R_{commit}. Thus, the average transaction response time, R, can be expressed as

$$R = R_{exec} + R_{commit}. \quad (14)$$

To estimate R_{exec} and R_{commit} for each of these policies, we have to estimate the number of I/O operations and number of CPU instructions executed in that phase. For each node, we can estimate R_{exec} as follows. Let R_{cpu} be the average time that a transaction spent at the processor, R_{io} be its average IO time, P_{cont}^i be the contention probability encountered by each

lock request to the ith partition and R_w be the average waiting time for a contended lock. Then R_{exec} can be expressed as

$$R_{exec} = R_{cpu} + R_{io} + L \sum_i \alpha_i P^i_{cont} R_w. \quad (15)$$

R_{io} is estimated from an infinite server model with the number of IO's determined form the appropriate buffer model. Hence, R_{io} can be expressed as $R_{io} = (N_{setup} + L(1 - H_X))T_{io}$, where N_{setup} is the number of initial I/O to load the transaction program, H_X is the buffer hit probability under coherency control policy X, and T_{io} is the time for an I/O operation. Without any loss of generality, we have assumed the network delay in our environment to be negligible. Note that if the network delay is not negligible it will increase the the transaction response time (due to global lock access) by the same amount under all coherency policies. Accounting for network delay for coherency messages will not increase the transaction response time, as the transaction can be committed even before the coherency messages are sent. However, the locks are retained until the acknowledgment for coherency messages are received.

As in [39] and [41], R_{cpu} is derived from an M/M/1 queueing model for the processor. Let M_{cpu} be the MIP's per processor. Let I_{cpu} be the total CPU demand (including policy overhead) per node. Then, CPU utilization, ρ, is given by $\rho = I_{cpu}/M_{cpu}$. Then, R_{cpu} for buffer coherency policy X is given by

$$R_{cpu} = \frac{\rho(I_{trx} + L(1 - H_X)I_{io})}{(1 - \rho)} \quad (16)$$

where I_{trx} is the number of instructions executed per transaction excluding the I/O and buffer coherency policy overhead, and I_{io} is the number of instructions executed per I/O operation.

Commit processing includes not only the writing of logs, propagation of updates to the disk, but also the overhead of coherency control. The number of updated pages written to disks, N_{upd}, can be expressed as $N_{upd} = L \sum_{i=1}^{P} \alpha_i p_{u_i}$. We assume the IO time for writing logs to be T_{log} and use I_{commit} to denote the number of instructions executed for writing the log and propagating the update to disks, which is independent of the coherency policies. Let I_{msg} be the number of instructions executed per message processing operation (SEND or RECEIVE). Processing an invalidation request has two components: 1) number of instructions executed to determine if the updated page is present in the local buffer (hash table look up etc.), I_{search}, and 2) number of instructions executed for purging the invalid page from the buffer, if present, I_{purge}. The CPU component for R_{commit} can be estimated in a similar way as (16).

Based on the CC model in [39] and [41], P^i_{cont} can be estimated as being proportional to the lock request rate to the ith partition and the average lock hold time while being inversely proportional to the size of the ith partition. We assume that a transaction can commit immediately after the updates and log record are written to disk (before the broadcast of the invalidation message). Assume that page accesses are spread uniformly across the transaction execution phase. Ignoring R_{commit}, it is shown in [39] that R_w would be roughly equal to $R_{exec}/3$ where the average lock hold time is $R_{exec}/2$. Define η to be $(R_{exec}/2)/(R_{exec}/2 + R_{commit})$. For CA and CAPN policies, extending the result in [39] to capture the fact that locks are held during commit phase, the average lock waiting time can be estimated as $\eta(R_{exec}/3 + R_{commit}) + (1 - \eta)R_{commit}/2$, where the first (respectively, second) term represents contention with transactions in the execution (respectively, commit) phase [9]. For BI, BU, SNI, and SNU policies, the locks are not released until the acknowledgment to invalidation or propagation messages from the other nodes are received. The average lock waiting time formula is similar to CA and CAPN with the exception that R_{commit} needs to be replaced by $R_{commit} + R_{inval}$, where R_{inval} is the time period needed to send out the invalidation messages and then wait for the messages to be processed and the acknowledgment returned. A more elaborate estimate for R_w can be found in [41].

1) Estimation of CPU Utilization: The CPU utilization, ρ, will be different for different policies and will depend on the value of I_{cpu}, the total CPU demand per processor. I_{cpu} is detailed below for each of the six buffer coherency policies.

CA Policy: This policy incurs no extra overhead as checking for invalidation is done along with lock request. Therefore,

$$I_{cpu} = \lambda\{I_{trx} + I_{commit} + L(1 - H_{ca})I_{io}\}. \quad (17)$$

BI Policy: Let N_{purge} be the total number of pages invalidated from a remote buffer due to the updates by a single transaction. N_{purge} can be expressed as $N_{purge} = L \sum_{i=1}^{P} \alpha_i p_{u_i} h_i$. Since, the system is symmetric the total CPU load per node can be estimated as the product of the transaction rate per node and the total CPU instructions executed on all nodes due to a single transaction. We will count CPU overhead in terms of the number of message processing operations (SEND or RECEIVE). The total number of message processing operations required per broadcast of invalidation is one send and $(N - 1)$ receives. The acknowledgment requires $(N - 1)$ pairs of send and receive. Hence, the total number of operations per broadcast is $(3N - 2)$. However, a broadcast is required only if the transaction updates one or more pages. The probability that transaction requires a broadcast, $p_{brdcast}$, is given by

$$p_{brdcast} = 1 - \prod_{i=1}^{P} (1 - p_{u_i})^{L\alpha_i}. \quad (18)$$

The average overhead in processing the invalidations (in all nodes) due to the updates by a single transaction is $(N - 1)(N_{upd} I_{search} + N_{purge} I_{purge})$. Hence,

$$\begin{aligned} I_{cpu} = \lambda\{&I_{trx} + I_{commit} + L(1 - H_{br})I_{io} \\ &+ (3N - 2)p_{brdcast} I_{msg} \\ &+ (N - 1)(N_{upd} I_{search} + N_{purge} I_{purge})\}. \end{aligned} \quad (19)$$

SNI Policy: This policy reduces the overhead in two ways over the BI policy. First, it sends fewer messages as it

notifies only the affected nodes. Second, it cuts down on invalidation requests by sending only the requests that result in buffer purge. Let N_{msg} be the total number of messages sent requesting invalidation per transaction.[2] That is to say N_{msg} is the number of nodes that have at least one page invalidated by the committing transaction. N_{msg} can be expressed as

$$N_{msg} = (N-1)\left[1 - \prod_{i=1}^{P}(1-h_i)^{L\alpha_i p_{u_i}}\right]. \quad (20)$$

Here, the second term in the outer product is the probability that at least one updated page is present in a particular node. Since, there are $(N-1)$ remote nodes, the average number of messages sent is given by (20). Now, sending invalidation request and receiving acknowledgment requires 2 pairs of message processing operations. Hence,

$$\begin{aligned} I_{cpu} = \lambda\{&I_{trx} + I_{commit} + L(1-H_{sni})I_{io} \\ &+ 4N_{msg}I_{msg} \\ &+ (N-1)N_{purge}(I_{search} + I_{purge})\}. \end{aligned} \quad (21)$$

BU Policy: This policy overhead is similar to that under the BI policy, except that some additional overhead is required to send long messages (copies of updated pages). Let I_{lmsg} be the additional CPU instructions required to send or receive each additional updated page. One other difference from the BI policy is that if a copy of an updated page is found in a remote buffer, the updated page is copied to rather than purged from the remote buffer. Let I_{copy} be the amount of overhead incurred to copy a page from the message buffer to the node database buffer. Therefore,

$$\begin{aligned} I_{cpu} = \lambda\{&I_{trx} + I_{commit} + L(1-H_{bu})I_{io} \\ &+ (3N-2)p_{brdcast}(I_{msg} + N_{upd}I_{lmsg}) \\ &+ (N-1)(N_{upd}I_{search} + N_{purge}I_{copy})\}. \end{aligned} \quad (22)$$

SNU Policy: This policy is similar to SN policy. Following the reasonings given in analyzing the BU policy, the overhead for this policy can be written as

$$\begin{aligned} I_{cpu} = \lambda\{&I_{trx} + I_{commit} + L(1-H_{snu})I_{io} \\ &+ 4N_{msg}I_{msg} \\ &+ (N-1)N_{purge}(2I_{lmsg} + I_{search} + I_{copy})\}. \end{aligned} \quad (23)$$

CAPN Policy: This policy incurs no extra message overhead, as the invalidation list is carried back with the reply to a lock request message. Hence, the only overhead is incurred by the processing of invalidation messages. Thus,

$$\begin{aligned} I_{cpu} = \lambda\{&I_{trx} + I_{commit} + L(1-H_{capn})I_{io} \\ &+ (N-1)N_{purge}(I_{search} + I_{purge})\}. \end{aligned} \quad (24)$$

[2]We assume no underlying network support for multicast operation (as discussed in [16]) and therefore, each node needs to be notified separately.

TABLE II
TRANSACTIONS AND SYSTEM PARAMETERS

System parameters	M_{cpu}	*20.0*
	T_{io}	*25ms*
	T_{log}	*5ms*
Workload parameters	D	*20,000*
	L	*10*
	λ	*35.0/sec.*
	p_u	*0.1*
CPU overheads (instructions)	I_{trx}	*350K*
	I_{io}	*3K*
	I_{msg}	*2.5K*
	I_{lmsg}	*2K*
	I_{search}	*0.5K*
	I_{purge}	*0.25K*
	I_{copy}	*0.5K*
I/O overheads	N_{setup}	*5*

V. VALIDATION AND RESULTS

In this section we will first validate our analyses of buffer models and integrated system models for the response times using a detailed simulation model for a multisystem data sharing environment. The simulation model captures all three components of the data sharing environment: buffer coherency policy, concurrency control protocol (locking), and the FCFS queueing discipline for the CPU. The details of various buffer coherency policies as described in earlier sections, accounting for CPU overheads, immediate versus delayed buffer invalidation as well as LRU replacement policy are implemented in our simulator. The simulation also explicitly keeps track of locks on data pages by transactions, and the validity of the contents of the buffers. In the case of a lock request leading to a data contention, the transaction is placed in a wait state until the lock is released by the transaction holding the lock. If a lock request leads to a deadlock, the transaction making the request is aborted and restarted after a back-off delay. Each rerun transaction makes the same references as its first run, and buffer hits result only if a copy is still in the buffer. The simulation process consists of two phases: initialization phase and data collection phase. The duration of the initialization phase was chosen such that the number of accesses to any buffer is at least an order of magnitude higher than the buffer size. In the graphs of the buffer hit probabilities and the mean response times shown in this section, confidence intervals were estimated using the method of batch means [27], and the 95% confidence interval was estimated to be within 2% of the mean. The parameter values used in the study are shown in Table II, unless otherwise specified. To study the effect of skewed access we assume that the database consists of two partitions: *Hot* and *Cold* sets. The update probabilities to the hot and cold pages are assumed to be the same and will be denoted as p_u.

Since the simulation for a larger buffer size and/or larger number of nodes is time consuming, we will mainly use the analytical model to study the difference in the buffer hit probabilities under the three different approaches (i.e., detection, notification, and update propagation) for various parameter ranges, e.g., skew in the access pattern, buffer size, invalidation rate, etc. In general, both the notification and de-

tection oriented approaches can suffer from buffer invalidation effect particularly for a larger number of nodes and higher update probability, while the buffer hit probability under the update propagation approach is the same as that of a single node system. Furthermore, the buffer hit probability under the notification oriented approach (BI and SNI) and CAPN is higher than that under the detection oriented approach (CA). This difference is very small under skewed access. However, the difference could be significant under uniform access for some specific range of system and workload parameters. On the other hand, both the CA and CAPN policies incur less policy overhead than the BI and SNI policies. The policy overheads for the update propagation approaches (BU and SNU) are the highest. Therefore, we will study the overall impact on the mean transaction response time under these coherency policies.

We will first study the performance of these policies under the homogeneous node access pattern, and in a later subsection will also study their performance under the disjoint hot set workload model.

A. Validation of Buffer Hit Probability and Response Times

Fig. 3 shows the validation of buffer hit probabilities of the CA policy varying the buffer size, for the skewed (80-20) and uniform access patterns, and for the systems consisting of 2 and 8 nodes based on the remaining parameter values given in Table II. In all cases, the match is excellent and the analysis and simulation results are indistinguishable. The buffer hit probabilities for the 8 node cases are smaller than those for the corresponding 2 node cases due to a larger amount of buffer invalidations. This difference is more pronounced for the skewed access pattern. Fig. 4 shows the corresponding validation of mean response times. Once again the match is quite good and the maximum difference between the simulation and the analysis is less than 2%. The response time curves follow the inverse trend of the buffer hit probabilities. The buffer model for the notification oriented approach and the equivalent model for the update propagation oriented approach (i.e., with no invalidation) are validated in earlier works [13], [10], [12] and found to have similar match as in the above model. The corresponding integrated system models for the the response times are also validated in those earlier works. Hence, we will use analytical models for the comparative study of the performances of these policies to avoid long simulations, especially for larger number of nodes and buffer sizes.

B. Comparison of Buffer Hit Probability

In this subsection we will compare the buffer hit probabilities under the different approaches for various parameter ranges, namely the skewed versus uniform access, the probability of update, the number of nodes, etc.. Fig. 5 compares the buffer hit probabilities of the notification (BI and SNI policies) and detection (CA policy) approaches, varying the number of nodes, for the skewed (80-20) and uniform access patterns. The CAPN policy has similar buffer hit probability as the notification approach. Also shown are the simulation

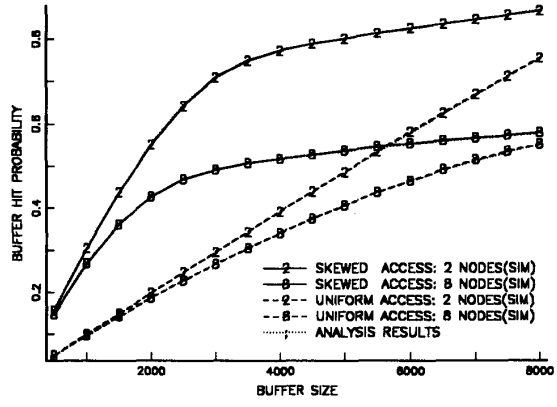


Fig. 3. Validation of buffer hit probability (CA policy, $p_u = 0.1$).

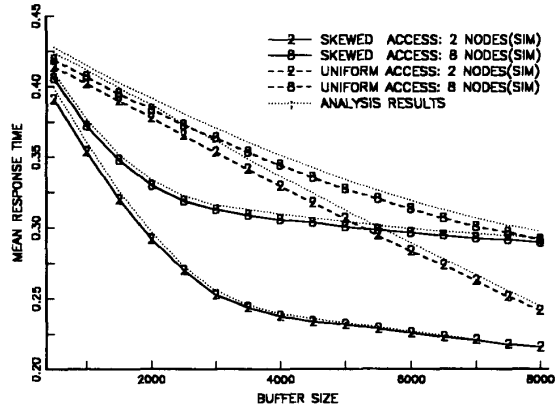


Fig. 4. Validation of mean response time (CA policy, $p_u = 0.1$).

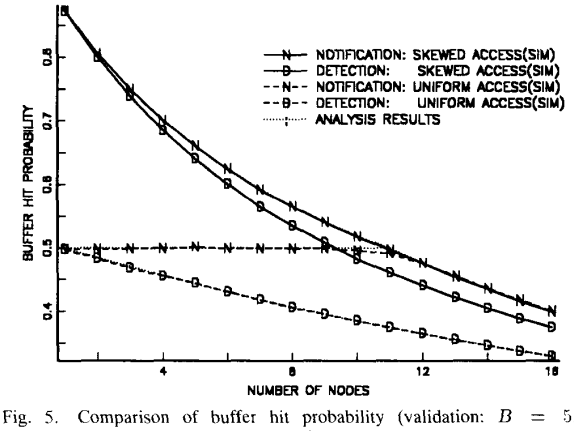


Fig. 5. Comparison of buffer hit probability (validation: $B = 5K$, $p_u = 0.1$).

results. Once again, the matches between simulation results and analytical predictions are excellent for all cases. (This should provide further confidence to the accuracy of the analyses.) There are several points to note here. First, under skewed access the difference in buffer hit probabilities under the two approaches is small for all range of nodes. However,

under a uniform access workload the difference can be much larger, where this difference first increases and then decreases with the increase in the number of nodes. The invalidation rate increases with the increase in the number of nodes and hence, under the detection approach, the number of invalidated pages present in the buffer also increases. This results in the decrease in the buffer hit probabilities under the detection approach for both access patterns. For uniform access, the buffer hit probability under the notification approach remains constant until some point (11 nodes in this case), and then follows the buffer hit probability curve for its skewed access counterpart. This is due to the fact that for a smaller number of nodes the miss rate is higher than the invalidation rate. Hence the buffer space available from page invalidations will be reused shortly afterwards and the buffer hit probability remains constant. Once the replacement rate becomes zero, the buffer hit probabilities for the pages of all partitions will be the same (for the reasons explained later in this section). Therefore, the buffer hit probabilities under both the skewed and uniform access patterns become the same for a larger number of nodes.

For the skewed access case the buffer hit probability decreases as the number of nodes increases under the notification approach. The reason is that as the invalidation rate increases, more cold pages are retained in the buffer. To better understand this phenomenon, we examine the buffer hit probabilities for each of the components (hot and cold) separately. Fig. 6 shows the buffer hit probabilities for the hot and cold pages under the notification and detection approaches for the skewed access (80-20) case. (Also shown are the simulation results and the match between simulation results and analytical predictions is again very close. This close match between simulation and analysis also holds for other measures considered in Figs. 7 and 8 but not shown for the clarity of presentation.) The buffer hit probabilities for the hot pages (represented by the top curve) are not distinguishable under the two approaches. This is because 1) the buffer size is large enough such that very few hot pages get replaced under either approach, 2) an early detection of invalidated pages under the notification only saves the cold pages from being replaced from the bottom of the LRU stack. Since most of the accesses and updates by a transaction lie in the hot set, most of the invalidations also go to the hot set. Hence, the buffer hit probability of the hot set drops rapidly. Under the detection approach the invalidated hot pages remain in the buffer until they are replaced or refreshed due to subsequent access to the same pages. Therefore, the total number of cold pages (valid or invalid) present in the buffer does not change as the number of nodes increases. The increased invalidation rate also reduces the number of valid cold pages (represented by the bottom curve), and hence, the buffer hit probability of the cold pages decreases, though by a small amount. Under the notification approach, as mentioned above, the invalidated hot pages are purged immediately, which results in fewer cold pages getting replaced from the bottom of LRU stack. Hence, the buffer hit probability for the cold pages (represented by the middle curve) actually increases with the increased invalidations of the hot pages. The buffer hit probability of the cold pages

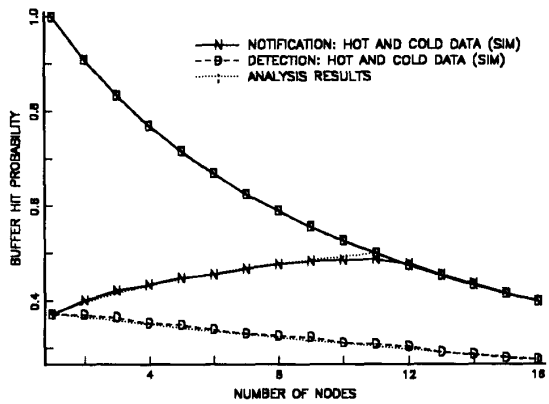


Fig. 6. Validation of buffer hit probability (access rule: 80-20, $B = 5K$, $p_u = 0.1$).

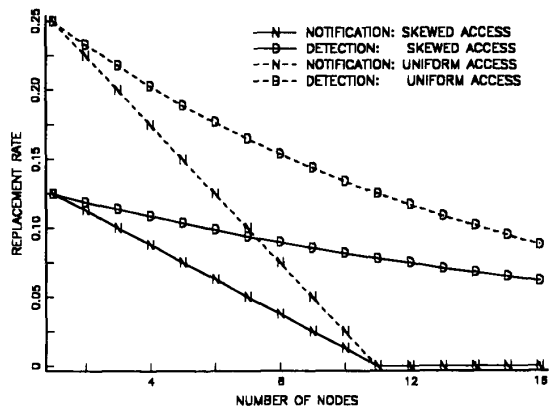


Fig. 7. Comparison of replacement rate of valid pages ($B = 5K$).

under the notification approach cannot increase indefinitely. It reaches its limit when the invalidation rate for the cold pages equals the rate at which they are brought back to the buffer, resulting in zero replacement rate as further explained in Fig. 7.

The replacement rates of the valid (hot + cold) pages are shown explicitly in Fig. 7 for both the skewed and uniform access patterns. Under the notification approach no valid page is replaced if there are buffer space available due to purging of invalid pages, while under the detection approach valid pages are replaced in the presence of undetected invalid pages. Hence, the detection approach can have a much higher replacement rate than the notification approach for a larger number of nodes, especially under a uniform access pattern. The maximum difference occurs at the point (11 nodes in this case) where the miss rate under the notification approach equals the invalidation rate. From this point on, the replacement rate under the notification approach drops to zero and the gap on replacement rates between the notification and detection approaches begins to reduce. Fig. 8 shows the probability that under the detection approach a buffer hit falls on an invalidated page in the buffer, for both the skewed and uniform access patterns. For skewed access, we further present the invalid hit

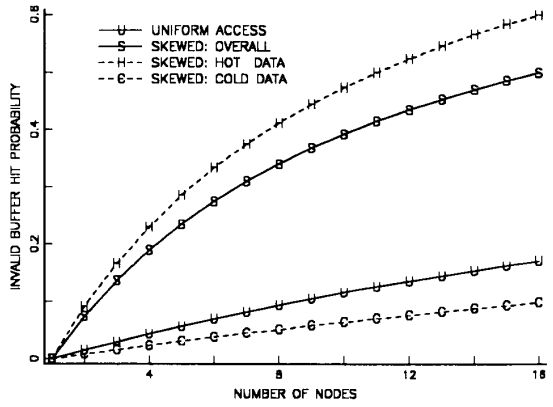


Fig. 8. Buffer hit probability of invalid pages ($B = 5K$).

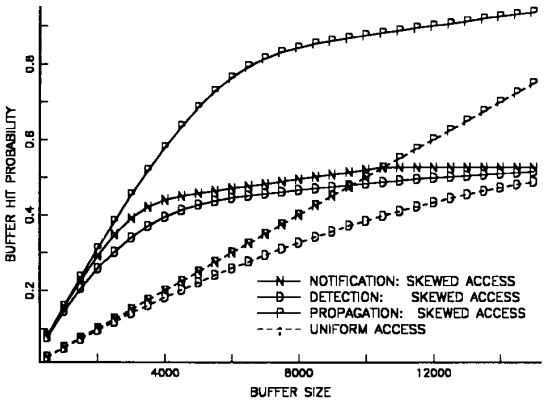


Fig. 9. Comparison of buffer hit probability ($N = 10$, $p_u = 0.1$).

probabilities to the hot and cold pages. Under skewed access, a much larger number of accesses on the hot pages lie on the invalidated pages.

Fig. 9 shows the buffer hit probabilities under all three approaches, the detection (CA policy), notification (BI and SNI policies and similarly for CAPN policy), and propagation (BU and SNU policies) approaches, as a function of buffer size for skewed (80-20) and uniform access patterns. Both the detection and notification approaches suffer from invalidation effect and their buffer hit probabilities can be significantly smaller than that under the propagation approach. For the uniform access pattern, the buffer hit probability under the notification approach increases with the buffer size and is exactly the same as that under the propagation approach until the point where the replacement rate is zero. Beyond this point, the additional buffer cannot be utilized by the notification approach as the miss rate is not fast enough to fill up the space vacated by invalidations and the buffer hit probability levels off. However, immediate propagation of updates can take advantage of this additional buffer. Hence, the buffer hit probability under the propagation approach continues to increase with the increase in buffer size. As pointed out earlier, under the detection approach a larger buffer is needed to achieve the same buffer hit probability. For a highly skewed access (80-20), the propagation approach does exceptionally well compared to the detection and notification approaches.

The difference in the buffer hit probabilities between the detection and notification approaches is small for the highly skewed access, and it is maximized under a uniform access pattern. The maximum difference occurs at a point when the replacement rate is zero. This break point, B_z, can be estimated by equating the miss rate to the invalidation rate, i.e., $B_z = D/(1 + (N - 1)p_u)$. At B_z, the miss rate is $\lambda L(1 - B_z/D)$ and the invalidation rate is $\lambda L(N-1)p_u B_z/D$. Note that this point is a function of both N and p_u. (Under skewed access, equating the miss rate to invalidation rate for each partition, we arrive at the same function for buffer hit probability. Therefore, the buffer hit probability is the same for all partitions beyond the break point.) This point occurs for different values of N in Fig. 10, where buffer hit probabilities for the detection and notification approaches are

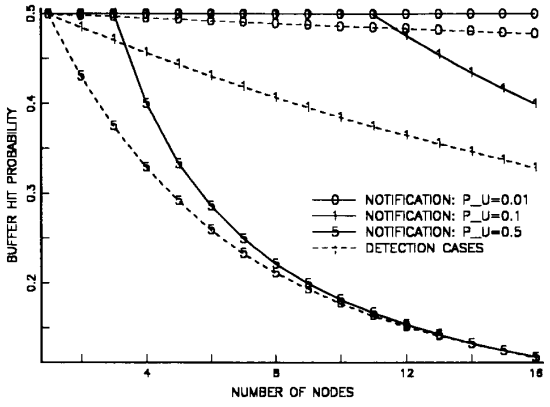


Fig. 10. Effect of update probability on the buffer hit probability ($B = 10K$, uniform access).

shown as a function of number of nodes for various update probabilities (0.01, 0.1, and 0.5), under the uniform access pattern.

In Fig. 11 we examine the effect of skewed access keeping the size of the hot set constant ($\beta_{hot} = 20\%$) and varying α_{hot}, the fraction of the accesses that goes to the hot set. The access pattern is varied from uniform ($\alpha_{hot} = 20\%$), to highly skewed ($\alpha_{hot} = 85\%$). The difference in the buffer hit probabilities between the detection and notification approaches as well as the absolute values of the buffer hit probabilities depend on the number of nodes, buffer sizes, and the probability of updates. For the 2 node case, when the invalidation rate is relatively small, the increased skewed access results in a higher buffer hit probabilities for both approaches, as frequently accessed hot pages are retained in the buffer [13]. Also, the difference in the buffer hit probabilities of the two approaches is quite small. However, for a larger number of nodes (10), this difference could be significant. The buffer hit probabilities for both approaches are smaller compared to the corresponding values for the 2 node case, due to the increased invalidation rate. Under uniform and less skewed access (i.e., for small values of α_{hot}) a larger degradation of the buffer hit probability is observed for the

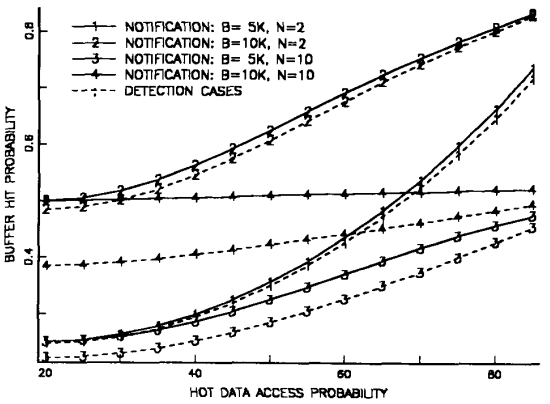


Fig. 11. Effect of skewed access on the buffer hit probability ($\beta_{hot} = 20\%$, $p_u = 0.1$).

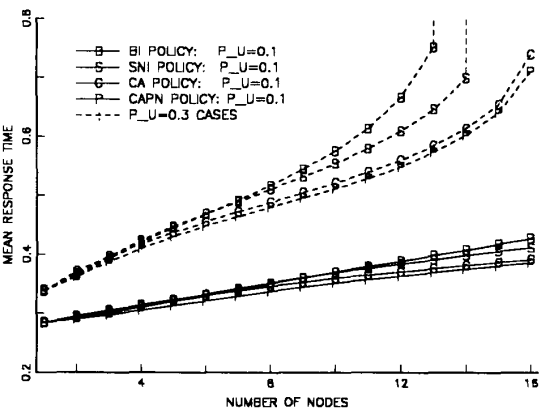


Fig. 12. Comparison of response times ($\lambda = 35$, $B = 5K$, access rule = 80-20).

detection approach especially for a larger buffer size (10K). With the increase in skewness, the invalidation rate on the hot pages also increases and a significantly lower number of (valid) hot pages can be retained in the buffer even with the notification oriented approach. Therefore, the gap in the buffer hit probabilities between the detection and notification approaches narrows.

C. Comparison of Transaction Response Time

We will first focus our attention to the policies that do not propagate the updates to the remote nodes as update propagation can incur a substantial amount of CPU overhead. Once the tradeoff in these policies are better understood we will then compare the performance of the best of these policies to that of the policies that propagate their updates. Figs. 12 and 13 compare the mean transaction response times under the BI, SNI, CA, and CAPN policies, varying the number of nodes, for the skewed (80-20) and uniform access patterns. In Fig. 12 the response times are shown for the probability of update values of 0.1 and 0.3. Recall from Fig. 5 that the buffer hit probabilities under skewed access are very close between the notification and detection oriented approaches. Hence, the differences in response times for the four policies in Fig. 12 are primarily due to the differences in CPU utilizations. For a smaller probability of update (0.1) fewer messages are sent and also fewer pages are invalidated. Hence, the response times under all three policies are very close. However, under a higher probability of update (0.3), these overheads increase for both the SNI and BI policies, especially for a larger number of nodes, which result in an increased CPU utilizations and response times. The number of messages under the BI policy is much higher than that under the SNI policy. The differences in response times are further magnified due to higher lock contention probability. As discussed in Section IV-B, the lock contention probability increases with the transaction rate and the lock holding time which grows with the transaction response time. As the number of nodes increases, not only the total transaction rate increases, but also the buffer hit probabilities decrease under all policies due to the increased

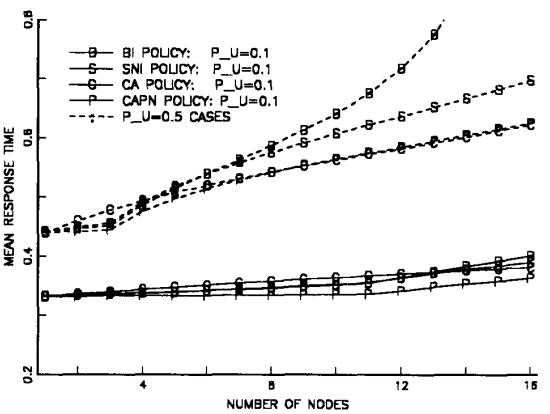


Fig. 13. Comparison of response times ($\lambda = 35, B = 10K$, access rule = uniform).

invalidation rates, and hence, the response times are inversely affected (i.e., increase). Both the CA and CAPN policies incur much less CPU overhead. The small difference in response times between the CA and CAPN policies are due to a small difference in buffer hit probabilities between the notification and detection oriented approaches.

The difference in buffer hit probabilities could be significant under a uniform access pattern for a specific combination of parameters like the number of nodes, buffer size and probability of update, as has been seen in the previous subsection. The resulting tradeoffs of buffer hit probability and policy overhead are shown in Fig. 13 for transaction response times versus the number of nodes under two different update probabilities (0.1 and 0.5). For a larger probability of update (0.5), this tradeoff is quite interesting. For a smaller number of nodes, when the message overheads for both the BI and SNI policies are small, they can perform better than the CA policy due to the difference in buffer hit probability. However, for a larger number of nodes the message overheads outweigh any advantage due to buffer hit probability. Note that CAPN that exploits the strengths of both the approaches (detection and notification) performs the best under all cases.

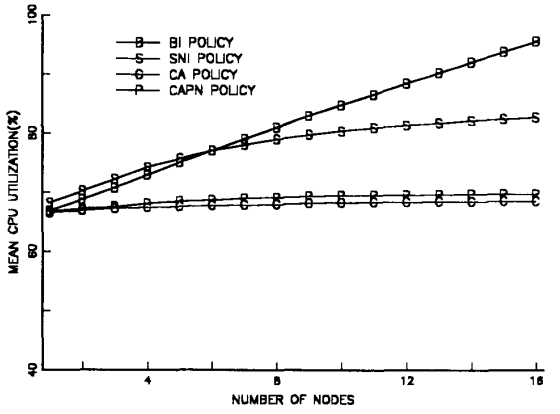


Fig. 14. Comparison of CPU utilization ($\lambda = 35$, $B = 10K$, access rule = uniform).

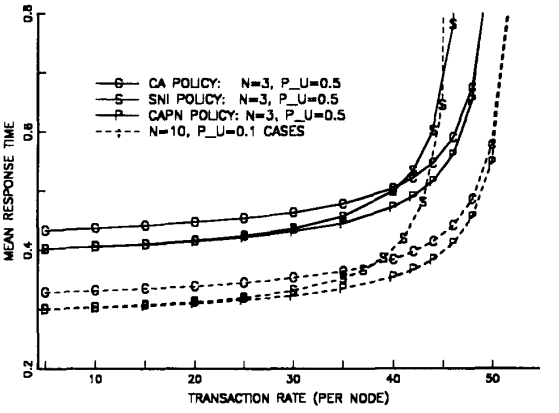


Fig. 15. Comparison of response times ($B = 10K$, access rule = uniform).

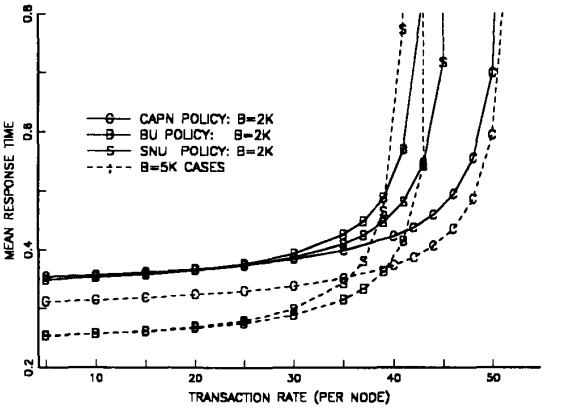


Fig. 16. Comparison of response times ($N = 10$, $p_u = 0.1$, access rule = 80-20).

Fig. 14 shows the corresponding CPU utilizations for $p_u = 0.5$. The CPU utilizations under both the CA and CAPN policies (represented by the two bottom curves) remain nearly constant with the increase in the number of nodes, as the number of I/O operations changes very little. (Note also the small difference in CPU utilizations between the CA and CAPN policies is due to a small additional overhead of purging the buffer under the CAPN policy.) The CPU utilizations under both the BI and SNI policies increase with the number of nodes. The increase in utilization is larger for the SNI policy compared to that of the BI policy for a smaller number of nodes while the reverse is true for a larger number of nodes. This implies that broadcast is preferable only for a small number of nodes, while substantial savings in overhead can be made for a larger number of nodes through selective notification. The cross-over point also depends on the buffer size, the probability of update and the access pattern.

Since the response time is a function of the CPU utilization, to better understand this tradeoff under the uniform access pattern the mean transaction response times are shown as a function of transaction rate in Fig. 15 for various combinations of the number of nodes, and the probability of update under CA, SNI, and CAPN. (Note that the BI policy is omitted as it incurs more overhead than the SNI policy except for a small number of nodes. Otherwise, its response time is similar to that of the SNI policy.) The notification oriented approach (SNI) can provide a lower response time due to higher buffer hit probability while the detection oriented approach (CA) that incurs a smaller overhead can provide a higher maximum throughput. Both the differences in response times and maximum throughputs depend on the buffer size, the access pattern, the probability of update and the number of nodes. CAPN is the preferred policy as it can take advantage of higher buffer hit probability through periodic notification, while paying no extra message overhead.

We now compare the performance of the CAPN policy to that of the policies that propagate the updated pages to remote nodes. Fig. 16 compares the response times of the CAPN, BU, and SNU policies as a function of transaction rate for buffer sizes of 2K and 5K under a skewed (80-20) access pattern. (Recall that in Fig. 9, the effect of propagation on buffer hit probability is most spectacular for the skewed access.) For a very small buffer size (2K), the buffer hit probability is the same under both the propagation and notification oriented approaches and hence, the response times are the same under the three policies for a lower transaction rate. However, the higher cost of propagation of updates implies lower maximum throughput for both the BU and SNU policies. Also, note that selective propagation is preferable compared to broadcasting the updates as very few invalidated pages are to be found in a small buffer. For a larger buffer size, the buffer hit probability under CAPN (or the notification oriented approach) is significantly smaller than that under the propagation oriented approach, as noted earlier in Fig. 9. Hence, for a lower transaction rate (i.e., for a lower CPU utilization) there is a significant difference in response times between the two approaches. However, the penalty in propagation of updates is a lower maximum throughput. Also note an interesting point about the maximum throughputs of the BU and SNU policies. The maximum throughput of the BU policy is higher than that of the SNU policy for the case of 5K buffer size, while the reverse is true for a smaller buffer size (2K). If invalidated

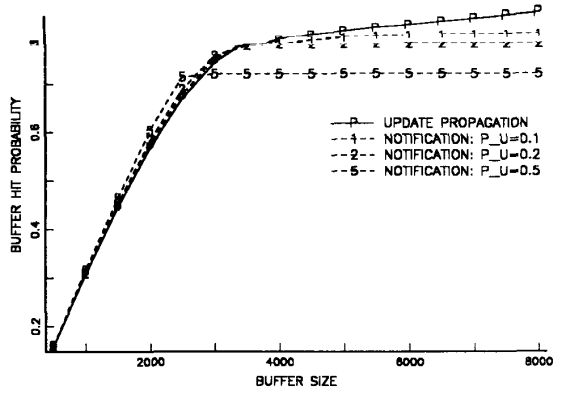


Fig. 17. Comparison of buffer hit probability ($N = 10$).

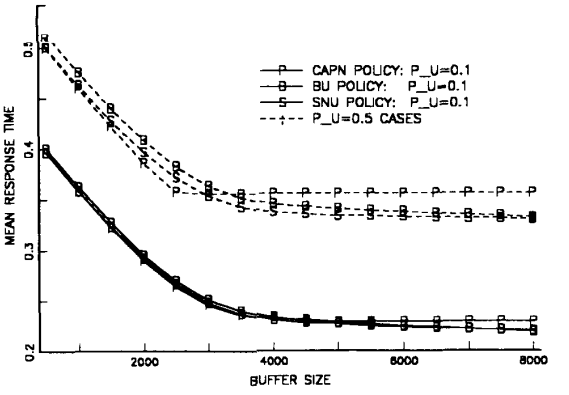


Fig. 18. Comparison of response times ($\lambda = 35$, $N = 10$, access rule = 80-10).

pages are more likely to be found in the buffer it is better to broadcast reducing the overhead of propagation.

D. Workload with Disjoint Hot Sets

We will now compare the performance of the three selected policies (CAPN, BU, and SNU) under a different workload where the sharing of data is not so strong. Each node has a distinct hot set of data which is accessed less frequently by the other nodes (see Section IV-A2). Hence, the invalidation problem will be less severe. Fig. 17 compares the buffer hit probabilities of the BU (similarly for SNU) and the CAPN policies for a 10 node system, and for different update probabilities (0.1, 0.2, and 0.5). (Note that the buffer hit probability for BU is not affected by the update probability.) Each node has a skewed access pattern of 80-10. That is to say that the database consists of 10 partitions where each partition is the hot set of one of the nodes receiving 80% of its references and a cold set to the rest of the nodes receiving 2.2% of their references. Under CAPN, the higher invalidation rate on the cold set increases the hot (and hence, overall) buffer hit probability by a small amount until the buffer size is large enough to contain the hot-set. Note that the reduction in buffer hit due to invalidation is modest even for a large number of nodes and a large probability of update. The resultant effect on the response time is shown in Fig. 18. The amounts of overhead incurred by the BU policy for all buffer sizes and the SNU policy for larger buffer sizes are significant (see Fig. 19), and hence, the maximum throughputs under these two policies can be significantly less compared to that under the CAPN policy. The difference in utilizations between the propagation oriented policies (BU and SNU) and CAPN increases with higher probability of update. Once again, the relative merit of the BU and the SNU policies depends on the probability that a copy of the updated page will be found in a remote buffer, which in turn depends on the buffer size and the access pattern.

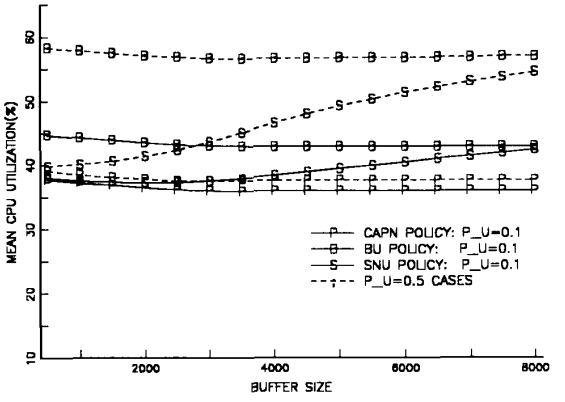


Fig. 19. Comparison of CPU utilization ($\lambda = 35$, $N = 10$, access rule = 80-10).

VI. Conclusion

In a data sharing environment, when a page gets updated by a node, copies of that page in other local buffers need to be invalidated. Different buffer coherency policies can be devised to address this issue. In this paper, we focus on analyzing the tradeoff of six different coherency control policies: check on access (CA), check on access with periodic notification (CAPN), selective notification (SNI), broadcast invalidation (BI), broadcast update (BU), and selective notification of update (SNU). These policies differ in their approaches to how and when invalidated pages are identified, and whether the updates are propagated. These differences result in different tradeoffs between CPU overhead and buffer hit probability. We developed analytic models to evaluate the buffer hit probability, CPU overhead and overall mean response time under these buffer coherency policies. We extended the approach developed in [13] for the BI policy to analyze other policies capturing the effect of skewed data access on the buffer hit probability and the invalidation or propagation overhead. In analyzing CA, we decomposed the buffer hit probability analysis into two components: 1) the probability of finding a page at a particular location in the LRU stack based on its frequency of access, and 2) the probability of the page being valid. The buffer hit probability is the sum of the product of these two components over all LRU stack positions in the buffer.

The BI, SNI, and CAPN policies provide the capability of early detection of buffer locations with obsolete pages for reclaim, while the BU and SNU policies propagate the updates, thus doing away with the invalidation effect on buffer hit probability. We found that the buffer hit probabilities under different policies can be strongly affected by the skewness in the data access pattern. For uniform access pattern, the effect of delayed detection of invalidated pages on buffer hit probability is most prominent. The reduction in buffer hit probability under the CA policy is proportional to the number of buffer locations containing obsolete data. For a highly skewed access pattern, if the buffer size is sufficiently large to cover the hot set, immediate purging of buffer locations with obsolete data leads to buffering of more cold pages. Hence, the improvement on the buffer hit probability using earlier detection, like the BI or SNI policy, can be considerably less. However, propagation of updates can lead to significant improvement on the buffer hit probability. Next we examine the additional CPU overhead, which is dominated by the messages for invalidation or propagation, incurred by the different policies. The number of messages incurred under the propagation oriented policy or (to a lesser extent) the notification oriented policy grows rapidly for a higher probability of update as the number of nodes increases while the CA or CAPN policy does not incur any message overhead. Although the SNI policy generally provides a way to reduce the number of messages compared to BI for a larger number of nodes, SNU may do a lot worse than BU due to the higher invalidation rate.

To summarize our observation on the mean transaction response time and throughput, under a skewed access pattern where the difference due to the buffer hit probabilities between the notification and detection oriented approaches is small, the policies that incur little overhead (CA and CAPN) can provide higher throughput and better mean response time than the notification oriented policies. The propagation oriented policies can improve the mean response time at the expense of throughput. However, under a uniform access pattern notification oriented policies may outperform the CA policy for a smaller number of nodes, and even for a larger number nodes if the probability of update is small. This is because under these situations the SNI and BI policies can take advantage of a higher buffer hit probability while incurring only a small to moderate amount of overhead, and the CAPN policy which combines the notification and detection oriented approaches would be able to provide the best performance.

REFERENCES

[1] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proc. 15th Int. Symp. Comput. Architecture*, Honolulu, HI, May 1988, pp. 280–289.

[2] J. Archibald and J. L. Baer, "Cache coherence protocols: Evaluation using a multiprocessor simulation model," *ACM Trans. Comput. Syst.*, vol. 4, no. 4, pp. 273–298, Nov. 1986.

[3] S. B. Behman, T. A. DeNatale, and R. W. Shomler, "Limited lock facility in a DASD control unit," Tech. Rep. TR 02.859, IBM General Products Division, San Jose, CA, Oct. 1979.

[4] M. Bellew, M. Hsu, and V-O. Tam, "Update propagation in distributed memory hierarchy," in *Proc. 6th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1990, pp. 521–528.

[5] J. K. Bennett, J. B. Carter, and W. Zwaenepoel, "Distributed shared memory based on type-specific memory coherence," in *Proc. 2nd ACM SIGPLAN Symp. Principles and Practices of Parallel Programming*, Seattle, WA, vol. 25, no. 3, Mar. 1990, pp. 168–176.

[6] A. Bhide, "An analysis of three transaction processing architectures," in *Proc. 14th Int. Conf. Very Large Databases*, Los Angeles, CA, Aug. 1988, pp. 339–350.

[7] M. J. Carey, M. J. Franklin, M. Livny, and E. J. Shekita, "Data caching tradeoffs in client-server DBMS architectures," in *Proc. ACM SIGMOD*, Denver, CO, May 1991, pp. 357–366.

[8] R. I. Casas and K. C. Sevcik, "A buffer management model for use in predicting overall database system performance," in *Proc. 5th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1989, pp. 463–469.

[9] B. Ciciani, D. M. Dias, and P. S. Yu, "Analysis of replication in distributed database systems," *IEEE Trans. Knowledge Data Eng.*, vol. 2, no. 2, pp. 247–261, June 1990.

[10] A. Dan, "Performance analysis of data sharing environments," Ph.D. dissertation, M.I.T. Press, Cambridge, Aug. 1992.

[11] A. Dan, D. M. Dias, and P. S. Yu, "Database buffer model for the data sharing environment," in *Proc. 6th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1990, pp. 538–544.

[12] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proc. ACM SIGMETRICS*, Denver, CO, (*Perform. Eval. Rev.*, vol. 18, no. 1), May 1990, pp. 143–152.

[13] A. Dan, D. M. Dias, and P. S. Yu, "The effect of skewed data access on buffer hits and data contention in a data sharing environment," in *Proc. 16th Int. Conf. Very Large Databases*, Brisbane, Australia, Aug. 1990, pp. 419–431.

[14] A. Dan, P. S. Yu, and J. Y. Chung, "Database access characterization for buffer hit prediction," in *Proc. 9th Int. Conf. Data Eng.*, Vienna, Austria, 1993.

[15] A. Dan and P. S. Yu, "Performance analysis of coherency control policies through lock retention," in *Proc. ACM SIGMOD*, San Diego, CA, June 1992, pp. 114–123.

[16] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LAN's," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85–110, May 1990.

[17] D. M. Dias, B. R. Iyer, and P. S. Yu, "Tradeoffs between coupling small and large processors for transaction processing," *IEEE Trans. Comput.*, vol. C-37, no. 3, pp. 310–320, Mar, 1988.

[18] D. M. Dias, B. R. Iyer, J. T. Robinson, and P. S. Yu, "Integrated concurrency-coherency controls for multisystem data sharing," *IEEE Trans. Software Eng.*, vol. 15, no. 4, pp. 437–448, Apr. 1989.

[19] M. Dubois and F. Y. Briggs, "Effects of cache coherency in multiprocessor," *IEEE Trans. Comput.*, vol. C-31, no. 11, pp. 1083–1099, Nov. 1982.

[20] S. J. Eggers and R. Katz, "Evaluating the performance of four snooping cache-coherency protocols," in *Proc. 16th Int. Symp. Comput. Architecture*, Jerusalem, Israel, May 1989, pp. 2–15.

[21] A. G. Greenberg and I. Mitrani, "Analysis of snooping caches," *Performance*, Dec. 1987.

[22] J. H. Howard *et al.*, "Scale and performance in a distributed file system," *ACM Trans. Comput. Syst.*, vol. 6, no. 1, pp. 51–81, Feb. 1988.

[23] Y. P. Hsu, "Performance evaluation of data sharing transaction processing systems," Master's thesis, Univ. Massachusetts, Amherst, Feb. 1988.

[24] J. P. Kearns and S. Defazio, "Diversity in database reference behavior," in *Proc. ACM SIGMETRICS*, (*Perform. Eval. Rev.*, vol. 17, no. 1), Berkeley, CA, pp. 11–19, May 1989.

[25] N. Kronenberg, H. Levy, and W. D. Strecker, "VAXcluster: A closely-coupled distributed system," *ACM Trans. Comput. Syst.*, vol. 4, no. 2, pp. 130–146, May 1986.

[26] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems," *ACM Trans. Comput. Syst.*, vol. 7, pp. 321–359, Nov. 1989.

[27] S. S. Lavenberg, Ed., *Computer Performance Modeling Handbook*. New York: Academic, 1983, pp. 93–95.

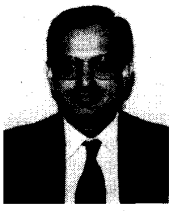
[28] C. Mohan and I. Narang, "Recovery and coherency control protocols for fast intersystem page transfer and fine granularity locking in a shared disks transaction environment," in *Proc. 17th Int. Conf. VLDB*, Barcelona, Spain, Sept. 1991, pp. 193–207.

[29] U. Ramachandran, M. Ahamad, and M. Y. A. Khalidi, "Coherence of distributed shared memory: Unifying synchronization and data transfer," in *Proc. 18th Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 1989, pp. II-160–II-169.

[30] E. Rahm, "Primary copy synchronization for DB-sharing," *Inform. Syst.*, vol. 11, no. 4, pp. 275–286, 1986.

[31] ——, "Empirical performance evaluation of concurrency and coherency control protocols for data sharing," IBM Res. Rep., RC 14325, 1988.

[32] J. T. Robinson, "A fast general-purpose hardware synchronization mechanism," *SIGMOD Rec.*, pp. 122–130, 1985.
[33] J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, "IMS/VS: An evolving system," *IBM Syst. J.*, vol. 21, no. 4, pp. 490–510, 1982.
[34] Y. C. Tay, N. Goodman, and R. Suri, "Locking performance in centralized databases," *ACM Trans. Database Syst.*, vol. 10, no. 4, pp. 415–462, Dec. 1985.
[35] J. Gray, Ed., *The Benchmark Handbook for Database and Transaction Processing Systems.* San Mateo, CA: Morgan Kaufmann, 1991.
[36] M. Vernon, E. D. Lazowska, and J. Zahorjan, "An accurate and efficient performance analysis technique for multiprocessor snooping cache-consistency protocols," in *Proc. 15th Int. Symp. Comput. Architecture,* Honolulu, HI, May 1988, pp. 308–315.
[37] K. Wilkinson and M. A. Neimat, "Maintaining consistency of client-cached data," in *Proc. 16th Very Large Database Conf.*, Brisbane, Australia, Aug. 1990, pp. 122–133.
[38] Q. Yang, L. N. Bhuyan, and B. Liu, "Analysis and comparison of cache coherence protocols for a packet-switched multiprocessor," *IEEE Trans. Comput.*, vol. C-38, no. 8, pp. 1143–1153, Aug. 1989.
[39] 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.
[40] P. S. Yu and D. M. Dias, "Concurrency control using locking with deferred blocking," in *Proc. 6th Int. Conf. Data Eng.*, Los Angeles, CA, Feb. 1990, pp. 30–36.
[41] P. S. Yu, D. M. Dias, and S. S. Lavenberg "On the analytic modeling of database concurrency control," IBM Res. Rep. RC 15386, Jan. 1990. *J. ACM*, to be published.
[42] P. S. Yu and D. M. Dias, "Analysis of hybrid concurrency control schemes for a high data contention environment," *IEEE Trans. Software Eng.*, vol. 18, no. 2, pp. 118–129, Feb. 1992.



Asit Dan (S'88–M'90) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, and the M.S. and Ph.D. degrees from the University of Massachusetts, Amherst, in computer science and computer engineering, respectively. His doctoral dissertation on "Performance Analysis of Data Sharing Environments" received an Honorable Mention in the 1991 ACM Doctoral Dissertation Competition.

Since 1990 he has been a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, NY. His current research interests include performance analysis, database systems, distributed systems, and workload analysis.

Recently, Dr. Dan received an Outstanding Innovation Award for his work on analytical modeling and comparison of various transaction processing architectures.

Philip S. Yu (S'76–M'78–SM'87–F'93), for a photograph and biography, see the January issue of this TRANSACTIONS, p. 86.