Client-Server Caching Revisited

Michael J. Franklin
Department of Computer Sciences
University of Wisconsin
Madison, WI 53706

Michael J. Carey
Department of Computer Sciences
University of Wisconsin
Madison, WI 53706

Abstract

The caching of data and/or locks at client workstations is an effective technique for improving the performance of a client-server database system. This paper extends an earlier performance study of client-server caching in several ways. The first is a re-examination of heuristics for deciding dynamically between propagating changes or invalidating remote copies of data pages in order to maintain cache consistency. The second is a study of the "Callback Locking" family of caching algorithms. These algorithms are of interest because they provide an alternative to the optimistic techniques used in the earlier study and because they have recently begun to find use in commercial systems. In addition, the performance of the caching algorithms is examined in light of current trends in processor and network speeds and in the presence of data contention.

1 INTRODUCTION

Object-Oriented Database Systems (OODBMS) are typically designed for use in networks of high-performance workstations and servers. As a result, most OODBMS products and research prototypes have adopted a variant of a *client-server* software architecture known as *data shipping*. In a data shipping system, client processes send requests for specific data items to servers, and servers respond with the requested items (and possibly others). Data shipping systems are particularly well-suited for use in workstation-based environments since they allow a DBMS to perform much of its work on client workstations, thus exploiting the workstations' processing and memory resources and offloading shared server machines. Data shipping systems can be categorized as *page servers*, in which clients and

servers interact using physical units of data such as pages or segments, and *object servers*, which send logical units of data such as objects or tuples between clients and servers [DeWitt, 1990] . Many recent systems (e.g., ObServer [Hornick, 1987], O2 [Deux, 1990], EXODUS [EXODUS Project, 1991, Franklin, 1992b], and ObjectStore [Lamb, 1991]) have adopted the page server approach due its relative simplicity and potential performance advantages compared to an object server approach.

Caching data and locks at client workstations is an important technique for improving the performance of client-server database systems. Data and locks can be cached both within a single transaction (*intra-transaction caching*) and across transaction boundaries (*inter-transaction caching*). Intra-transaction caching is easily implemented; it requires only that a client can manage its own buffer pool and can keep track of the locks it has obtained. Due to this simplicity, all of the algorithms investigated in this study perform intra-transaction caching of both data and locks. On the other hand, inter-transaction data caching requires additional mechanisms to ensure the consistency of cached data pages, and inter-transaction lock caching requires full-function lock management on clients and protocols for distributed lock management. In this study, we concentrate on the performance tradeoffs of inter-transaction caching. For the remainder of the paper, use the term "caching" to refer exclusively to inter-transaction caching at client workstations. Also, for concreteness, this study concentrates page server architectures; however, many of the results are also applicable to object server systems.

1.1 PREVIOUS WORK

Several previous studies have shown that the caching of data and locks can have substantial performance benefits [Wilkinson, 1990, Carey, 1991a, Wang, 1991]. These benefits can result from: 1) reducing reliance on the server, thus offloading a potential bottleneck and reducing communication, 2) allowing better utilization of the CPU and memory resources that are available on clients, and 3) increasing the scalability of the system in terms of the impact of adding client workstations. However, caching also has the potential to degrade performance due to the overhead of maintaining cache consistency. This overhead can manifest itself in several forms: 1) increased communication, 2) increased load on the server CPU, 3) additional path length, and 4) extra load placed on clients. Also, some algorithms may postpone the discovery of data conflicts, resulting in increased transaction abort rates.

In an earlier paper [Carey, 1991a] we investigated the performance implications of inter-transaction data and lock caching. In that paper, five locking-based algorithms were compared: an algorithm that performed no caching, one that cached data (but not locks), and three variants of an *Optimistic Two-Phase Locking* (O2PL) algorithm that allowed caching of data pages and an optimistic form of lock caching. With O2PL, locks are obtained only locally at clients during transaction execution and then, prior to commit, locks on all copies of updated pages are obtained at remote sites in order to ensure consistency. This form of lock caching is optimistic in the sense that a local lock at one site does not guarantee the absence of conflicting local locks at other sites. The three O2PL-based algorithms differed in the actions that they performed at the remote sites once locks were obtained. One variant, called O2PL-Invalidate (O2PL-I), always removed the copies from the remote sites. The second variant, O2PL-Propagate (O2PL-P), always sent new copies of the

updated pages to the remote sites, thereby preserving replication. The third variant, O2PL-Dynamic (O2PL-D), was an adaptive algorithm that used a simple heuristic to choose between invalidation and propagation on a copy-by-copy basis.

In the previous study, a number of conclusions were reached about the relative merits of these alternative caching algorithms. These include the following: 1) All of the caching algorithms were shown to have much higher performance than the non-caching algorithm in most of the workloads examined. 2) The optimistic caching of locks in addition to the caching of data was found to improve performance except in a workload with extremely high data contention and in some cases where the O2PL-P algorithm performed excessive update propagation. 3) The invalidation-based version of O2PL (O2PL-I) typically had the highest throughput. 4) The propagate version of O2PL (O2PL-P) was found to have significant performance advantages for certain types of data sharing, but also displayed a high degree of volatility, especially with regard to client buffer pool sizes. 5) The dynamic O2PL algorithm (O2PL-D) was seen to approach, but not quite equal, the performance of the better of the static O2PL algorithms over a wide range of workloads.

1.2 EXTENSIONS TO THE PREVIOUS STUDY

In this paper, we extend the results of the previous study in the following ways:

1. *A Better Adaptive Heuristic*: As stated above, the O2PL-Dynamic algorithm usually approached, but never met or exceeded the performance level of the better of the two static O2PL algorithms for a given workload. Thus, the first issue addressed here is to find a better heuristic for the adaptive algorithm.

2. *Callback Locking*: An alternative method for maintaining cache consistency is *Callback Locking* [Howard, 1988, Lamb, 1991, Wang, 1991]. Callback locking allows clients to cache both data pages and locks, but unlike O2PL, lock caching is not optimistic — sites are not allowed to simultaneously hold conflicting locks.

3. *System Parameter and Workload Changes*: Current trends in hardware technology have the potential to shift the performance bottlenecks in the system and may alter the comparative advantages of the caching algorithms. For example, CPU speeds are increasing dramatically while disk speeds are not; network technology has been improving, but most installed networks have yet to catch up. In this study, we examine the effects of these changes by increasing the speeds of the client and server CPUs compared to the earlier study, and by examining the effect of two different network bandwidths. We also use an additional workload to help examine the performance of the caching algorithms in the presence of high data contention.

The remainder of the paper is structured as follows: Section 2 describes the three types of cache consistency algorithms that are investigated in this study. Section 3 describes the simulation model and workloads. Section 4 presents the experiments and results. Section 5 discusses related work, and Section 6 presents conclusions.

2 OVERVIEW OF CACHING ALGORITHMS

The algorithms of this study fall into three families: Server-based 2PL, Optimistic 2PL (O2PL), and Callback Locking. These are described in the following sections.

2.1 SERVER-BASED TWO-PHASE LOCKING

Server-based 2PL schemes are derived from *primary-copy* concurrency control algorithms, with the server's copy of each page being treated as the primary copy of that page. Client transactions must obtain the proper lock from the server before they are allowed to access a data item. Clients are not allowed to cache locks across transaction boundaries. In this study we use a variant called Caching 2PL (*C2PL*) which allows *data pages* to be cached at clients across transaction boundaries. Consistency is maintained using a "check-on-access" policy. When a transaction requests a read lock for a page that is cached at its client, it sends the Log Sequence Number (LSN) found on its copy of the page along with the lock request. When the server responds to the lock request, it includes an up-to-date copy of the page along with the response if it determines that the site's copy is no longer current. In C2PL, deadlock detection is performed exclusively at the server. Deadlocks are resolved by aborting the youngest transaction involved in the deadlock. An algorithm similar to C2PL is currently used in the EXODUS storage manager.

2.2 OPTIMISTIC TWO-PHASE LOCKING (O2PL)

The O2PL schemes allow inter-transaction caching of data pages and and an optimistic form of lock caching. They are based on a *read-one, write-all* concurrency control protocol for replicated data in distributed databases, which was studied in [Carey, 1991b]. The O2PL algorithms are "optimistic" in the sense that they defer the detection of conflicts among locks cached at multiple sites until transaction commit time. In these algorithms, each client has its own local lock manager from which the proper lock must be obtained before a transaction can access a data item at that client. A non-two-phase read lock (i.e., latch) is obtained briefly at the server when a data item is in the process of being prepared for shipment to a client, thus ensuring that the client is given a transaction-consistent copy of the page. The server is responsible for keeping track of where pages are cached in the system. Clients inform the server when they drop a page from their buffer pool by piggybacking that information on the next message that they send to the server. Thus, the server's copy information is conservative, as there may be some delay before the server learns that a page is no longer cached at a client.

When an updating transaction is ready to enter its commit phase, it sends a message to the server containing a copy of each page that it has updated. The server then acquires exclusive locks on these pages on behalf of the transaction. Once these locks have been acquired at the server, the server sends a message to each client that has cached copies of any of the updated pages. These remote clients obtain exclusive locks on their local copies of the updated pages on behalf of the committing transaction. Once all the required locks have been obtained, variant-specific actions are taken. O2PL-Invalidate (*O2PL-I*) invalidates the remote cached copies of data pages, while O2PL-Propagate (*O2PL-P*) propagates the new values of data items to remote caching sites; O2PL-Dynamic (*O2PL-D*) chooses between invalidation and propagation on a per-copy basis. All sites that propagate one or more new page values must participate in a two-phase commit protocol with the server to ensure atomicity. Invalidation does not require a two-phase commit, as it does not result in the updating of any data that remains valid at the site.

With O2PL, each client maintains a local waits-for graph which is used to detect deadlocks that are local to the client. Global deadlocks are detected using a centralized algorithm *a la* [Stonebraker, 1979]. The server periodically requests local waits-for graphs from the clients and combines them to build a global waits-for graph. Deadlocks involving consistency actions can be detected early. When a conflict is detected between exclusive locks for two consistency actions or between a consistency action lock and a write lock, it is known that a deadlock has occurred or will shortly occur, so the deadlock can be resolved immediately [Carey, 1991b]. As in the server-based case, deadlocks are resolved by aborting the youngest transaction.

2.3 CALLBACK LOCKING

The third family of caching algorithms studied is Callback Locking. Callback Locking allows caching of data pages and non-optimistic caching of locks. In contrast to the O2PL algorithms, with Callback Locking clients must obtain a lock from the server immediately (rather than at commit time) prior to accessing a data page, if they don't have the proper lock cached locally. When a client requests a lock that conflicts with one or more locks that are currently cached at other clients, the server "calls back" the conflicting locks by sending requests to the sites which have those locks cached. The lock request is granted only when the server has determined that all conflicting locks have been released. As a result, transactions do not need to perform consistency maintenance actions during the commit phase. In this study, we analyze two Callback Locking variants: Callback-Read(CB-Read), which allows inter-transaction caching of read locks only, and Callback-All (CB-All), which allows inter-transaction caching of read locks and write locks.

As in the O2PL algorithms, the server keeps track of which sites have copies of pages in their cache. CB-Read, which caches only read locks works as follows. When a request for a write lock on a page arrives at the server, the server issues callback requests to all sites (except the requester) that have a cached copy of the page. At a client, such a callback request is treated as a request for an exclusive lock on the specified page. If the request can not be granted immediately (due to a lock conflict), the client responds to the server saying that the page is currently in use. When the callback request is granted at the client, the page is removed from the client's buffer and an acknowledgement message is sent to the server. When all callbacks have been acknowledged to the server, the server grants a write lock on the page to the requesting client. Any subsequent read or write lock requests for the page will be blocked at the server until the write lock is released by the holding transaction. At the end of the transaction, the client sends copies of the updated pages to the server and releases its write locks; retaining copies of the pages (and hence, implicit read locks on the pages) in its cache.

The CB-All algorithm works similarly to CB-Read, except that write locks are kept at the clients rather than at the server and are not released at the end of a transaction. The server's copy information is augmented to allow a copy at a site to be designated as an *exclusive copy*, and clients also keep track of which of their pages are cached exclusively. If a read request for a page arrives at the server and an exclusive copy of the page is currently held at some site, a downgrade request is sent to that site. A downgrade request is similar to a callback request, but rather than removing the page from its buffer, the client simply notes that it no longer

has an exclusive copy of the page; in effect it downgrades its cached write lock to a read lock. Non-exclusive copies are treated as in the CB-Read algorithm. As in CB-Read, clients send copies of pages dirtied by a transaction to the server when the transaction commits. However, in CB-All this is done only to to simplify recovery, as no other sites can access a page while it is exclusively cached at another site.

Callback algorithms were originally introduced to maintain cache consistency in distributed file systems such as Andrew [Howard, 1988] and Sprite [Nelson, 1988], which provide a weaker form of consistency than that required by database systems. More recently, a Callback Locking algorithm that provides transaction serializability has been employed in the ObjectStore OODBMS. An algorithm similar to CB-Read was studied in [Wang, 1991]. That algorithm differed from CB-Read with regard to deadlock detection — it considered all cached locks to be in-use for the purpose of computing the waits-for graph at the server. This could cause aborts due to phantom deadlocks; especially with large client caches. To avoid this problem, CB-Read (and CB-All) adopt a convention from ObjectStore: copy sites always respond to a callback request immediately, even if the page is currently in use at the site. This allows deadlock detection to be performed with accurate information.

2.4 SUMMARY OF THE ALGORITHMS

We now briefly summarize the three types of algorithms that are investigated in this study. All of the algorithms allow caching of data pages. C2PL does not allow caching of locks — it uses check-on-access to ensure the consistency of accessed data. The O2PL algorithms allow an optimistic form of lock caching: locks are obtained locally at clients during transaction execution, deferring global acquisition of locks until the commit phase. The three O2PL variants use invalidation and/or propagation to maintain consistency. The Callback Locking algorithms allow true inter-transaction caching of locks, but they require non-cached locks to be obtained at the server immediately during transaction execution. The Callback Locking algorithms of this study are invalidation-based.

3 A CLIENT-SERVER CACHING MODEL

3.1 THE SYSTEM MODEL[1]

Figure 1 shows the structure of our simulation model, which was constructed using the DeNet discrete event simulation language [Livny, 1988]. It consists of components that model diskless client workstations and a server machine (with disks) that are connected over a simple network. Each client site consists of a *Buffer Manager* that uses an LRU page replacement policy, a *Concurrency Control Manager* that is used as either as a simple lock cache or as a full-function lock manager (depending on the cache consistency algorithm in use), a *Resource Manager* that provides CPU service and access to the network, and a *Client Manager* that coordinates the execution of transactions at the client. Each client also has a module called the *Transaction Source* which submits transactions to the client according to the workload model described in the following section. Transactions are represented as page

[1] A more detailed description of the model can be found in [Carey, 1991a].

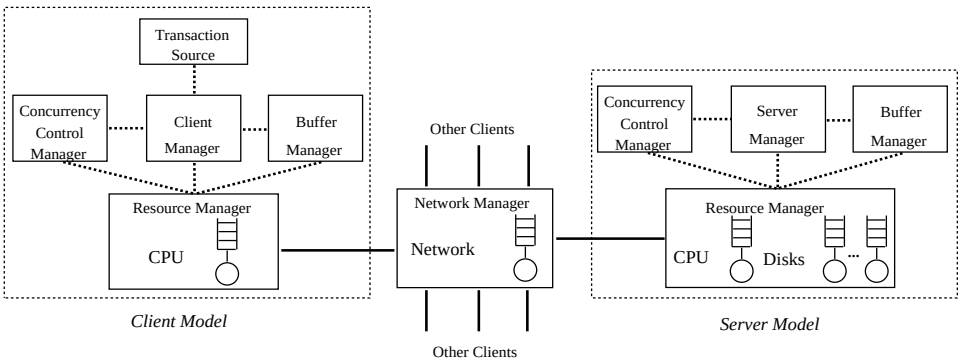

Figure 1: Model of a Client-Server DBMS

Parameter	Meaning	Setting
ClientCPU	Instruction rate of client CPU	15 MIPS
ServerCPU	Instruction rate of server CPU	30 MIPS
ClientBufSize	Per-client buffer size	5% or 25% of DB size
ServerBufSize	Server buffer size	50% of DB size
ServerDisks	Number of disks at server	2 disks
MinDiskTime	Minimum disk access time	10 millisecond
MaxDiskTime	Maximum disk access time	30 milliseconds
NetworkBandwidth	Network bandwidth	8 or 80 Mbits per sec
PageSize	Size of a page	4,096 bytes
DatabaseSize	Size of database in pages	1250 (5 MBytes)
NumClients	Number of client workstations	1 to 25
FixedMsgInst	Fixed no. of instructions per message	20,000 instructions
PerByteMsgInst	No. of addl. instructions per msg. byte	2.44 instructions
ControlMsgSize	Size in bytes of a control message	256 bytes
LockInst	No. of instructions per lock/unlock pair	300 instructions
RegisterCopyInst	No. of inst. to register/unregister a copy	300 instructions
DiskOverheadInst	CPU Overhead for performing disk I/O	5000 instructions
DeadlockInterval	Global deadlock detection frequency	1 second (if needed)

Table 1: System and Overhead Parameters

reference strings and are submitted to the client one at a time; upon completion of a transaction, the source waits for a specified think time and then submits a new transaction. When a transaction aborts, it is resubmitted with the same page reference string. The number of client machines is a parameter to the model. The server is modeled similarly to the clients, but with the following differences: the Resource Manager manages disks as well as a CPU, the Concurrency Control Manager has the ability to store information about the location of page copies in the system and also manages locks, there is a *Server Manager* component that coordinates the operation of the server (analogous to the client's Client Manager), and there is no Transaction Source module (since all transactions originate at client workstations).

Table 1 describes the parameters that are used to specify the resources and overheads of the system and shows the settings used in this study. The simulated CPUs of the system are managed using a two-level priority scheme. System CPU requests, such as those for message and disk handling, are given priority over

Parameter	Meaning
TransSize	Mean number of pages accessed per transaction
HotBounds	Page bounds of hot range
ColdBounds	Page bounds of cold range
HotAccProb	Prob. of accessing a page in the hot range
HotWrtProb	Prob. of writing to a page in the hot range
ColdWrtProb	Prob. of writing to a page in the cold range
PerPageInst	Mean no. of CPU instructions per page on read (doubled on write)
ThinkTime	Mean think time between client transactions

Table 2: Workload Parameter Meanings

user (transaction) requests. System requests are handled using a FIFO queueing discipline, while a processor-sharing discipline is employed for user requests. Each disk has a FIFO queue of requests; the disk used to service a particular request is chosen uniformly from among all the disks at the server. The disk access time is drawn from a uniform distribution between a specified minimum and maximum. A very simple network model is used in the simulator's *Network Manager* component; the network is modeled as a FIFO server with a specified bandwidth. We did not model the details of the operation of a specific type of network (e.g., Ethernet, token ring, etc.). Rather, the approach we took was to separate the CPU costs of messages from the on-the-wire costs of messages, and to allow the on-the-wire message costs to be adjusted using the bandwidth parameter. The CPU cost for managing the protocol for a send or a receive of a message is modeled as a fixed number of instructions per message plus a charge per message byte.

3.2 WORKLOADS

Our simulation model provides a simple but flexible mechanism for describing workloads. The access pattern for each client can be specified separately using the parameters shown in Table 2. Transactions are represented as a string of page access requests in which some accesses are for reads and others are for writes. Two ranges of database pages can be specified: a hot range and a cold range. The probability of a page access being to a page in the hot range is specified; the remainder of the accesses are directed to cold range pages. For both ranges, the probability that an access to a page in the range will be a write access (in addition to a read access) is specified. The parameters also allow the specification of an average number of instructions to be performed at the client for each page access, once the proper lock has been obtained. This number is doubled for write accesses.

Table 3 summarizes the workloads that are used in this study. The HOTCOLD workload has a high degree of locality per client and a moderate amount of sharing and data contention among clients. The FEED workload represents a situation such as a stock quotation system in which one site produces data while the other sites consume it. UNIFORM is a low-locality, moderate write probability workload which is used to examine the consistency algorithms in a case where caching is not expected to pay off significantly. Finally, HICON is a workload with varying degrees of data contention. It is similar to skewed workloads that are often used to study shared-disk transaction processing systems, and it is introduced here to investigate the effects of data contention on the various algorithms.

Parameter	HOTCOLD	FEED	UNIFORM	HICON
TransSize	20 pages	5 pages	20 pages	20 pages
HotBounds	p to p+49, p = 50(n-1)+1	1 to 50	-	1 to 250
ColdBounds	rest of DB	rest of DB	whole DB	rest of DB
HotAccProb	0.8	0.8	-	0.8
ColdAccProb	0.2	0.2	1.0	0.2
HotWrtProb	0.2	1.0/0.0	-	0.0 to 0.5
ColdWrtProb	0.2	0.0/0.0	0.2	0.2
PerPageInst	30,000	30,000	30,000	30,000
ThinkTime	0	0	0	0

Table 3: Workload Parameter Settings for Client n

4 EXPERIMENTS AND RESULTS

In this section we present the results of the performance studies of the new heuristic for the adaptive O2PL algorithm and of the Callback Locking algorithms. For most experiments the main metric presented is throughput, measured in transactions per second. Where necessary, auxiliary performance measures are also discussed. An additional metric that is used in several of the experiments is the number of messages sent per committed transactions. This metric is computed by taking the total number of messages sent during the simulation runs (by clients and the server) and dividing by the number transaction commits that occur during the simulation run. Results are presented for two different network bandwidths called "slow" (8 Mbits/sec) and "fast" (80 Mbits/sec); these correspond to slightly discounted bandwidths of Ethernet and FDDI technology, respectively.

4.1 A NEW ADAPTIVE HEURISTIC

As described earlier, the heuristic for the dynamic O2PL algorithm used in [Carey, 1991a] generally performed well, but was never able to match the performance of the better of the other two O2PL algorithms for a given workload (except in a workload with no data contention, where all O2PL algorithms performed identically). The original O2PL-D algorithm uses a simple heuristic to determine whether to use invalidation or propagation in order to ensure that a site does not retain an out-of-date copy of a page. It initially propagates updates, invalidating copies on a subsequent consistency operation if it detects that the preceding page propagation was wasted. Specifically, O2PL-D will propagate a new copy of a page if both: *1) the page is resident at the site when the consistency operation is attempted*, and *2) if the page was previously propagated to the site, then the page has been re-accessed since that propagation.* In the initial study, it was found that the algorithm's willingness to err on the side of propagation resulted in its performance being somewhat lower than that of O2PL-I in most cases. The new heuristic, therefore, was designed to instead err on the side of invalidation if a mistake was to be made. In the new dynamic algorithm, O2PL-ND (for "New Dynamic"), an updated copy of a page will be propagated to a site only if conditions 1 and 2 of O2PL-D hold plus *3) the page was previously invalidated at that site and that invalidation was a mistake.* The new condition ensures that O2PL-ND will invalidate a page at a site at least once before propagating it to that site.

In order to implement the new condition, it is necessary to have a mechanism for determining that an invalidation was a mistake. This is not straightforward since invalidating a page removes the page and its buffer control information from the site, leaving no place to store information about the invalidation. To solve this problem we use a structure called the *invalidate window*. The invalidate window is a list of the last n distinct pages that have been invalidated at the site. The window size, n, is a parameter of the O2PL-ND algorithm. When a page is invalidated at a site, its page number is placed at the front of the invalidate window on that site. If the window is full and the page does not already appear in the window, then the entry at the end is pushed out of the window to make room for the new entry. If the page already appears in the window, it is simply moved from its present location to the front. When a transaction's page access request results in a page being brought into the site's buffer pool, the invalidate window is checked and if the page appears in the window, the page is marked as having had a mistaken invalidation applied to it.[2] The page remains marked as long as it resides in the client's buffer.

Except for the different heuristic, the O2PL-ND algorithm works much like O2PL-D. When a consistency action request arrives at a client, the client obtains exclusive locks on its copies of the affected pages. The client then checks the remaining two conditions for propagation for each page and if both conditions are met the client will decide to receive a new copy of the page. The client retains its locks on those pages it has chosen to propagate and invalidates (also releasing its locks on) the other pages affected by the consistency request. The client then sends a message to inform the server of its decision(s). If all decisions were to invalidate, then the client is finished with the consistency action. Otherwise, this message acts as the acknowledgement in the first phase of a two-phase commit protocol. When the server has heard from all involved clients, it sends copies of the necessary pages to those sites that requested them. This message serves as the second phase of the commit protocol. Upon receipt of the new page copies, the clients install them in their buffer pools and release the locks on those pages. The receipt of a propagated page copy at a client does not affect the LRU status of the page at that site.

The performance of the new heuristic was examined using the HOTCOLD, FEED, and UNIFORM workloads. In all of the experiments described here, the O2PL-ND algorithm was run with a window size of 20 pages. An early concern with the new heuristic was its potential sensitivity to the window size. A series of tests using the parameters from [Carey, 1991a] found that in all cases but one, the algorithm was insensitive to the window size within a range of 10 to 100 pages (0.8% to 8% of the database size). The exception case and the reasons for the insensitivity of the algorithm in the other cases will be discussed in the following sections.

4.1.1 Experiment 1 : The HOTCOLD Workload

Figure 2 shows the throughput for the HOTCOLD workload using the slow network and a client buffer size of 5% of the database size (62 pages). This case shows a clear example of how the new heuristic of O2PL-ND can improve performance over

[2] Marking the page at this time limits the algorithm's sensitivity to the window size. If the window was checked at consistency maintenance time, then a small window size could result in useful pages being invalidated because they have been "pushed out of the window" even while they were being used at the client.

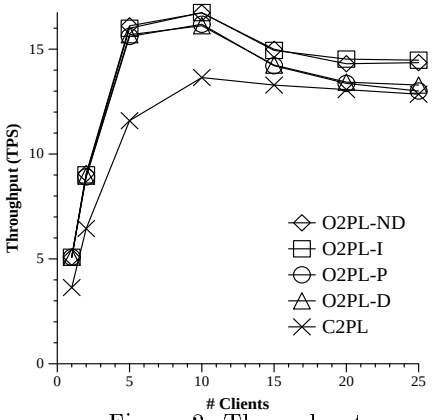

Figure 2: Throughput
(HOTCOLD, 5% Cli Buffers, Slow Net)

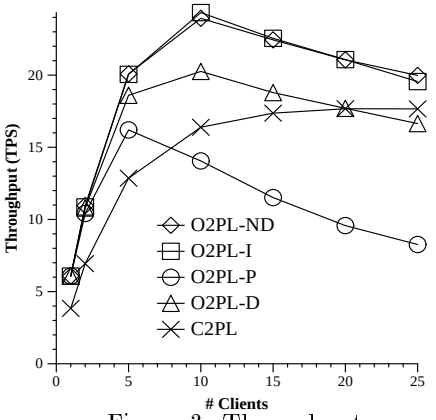

Figure 3: Throughput
(HOTCOLD, 25% Cli Buffers, Slow Net)

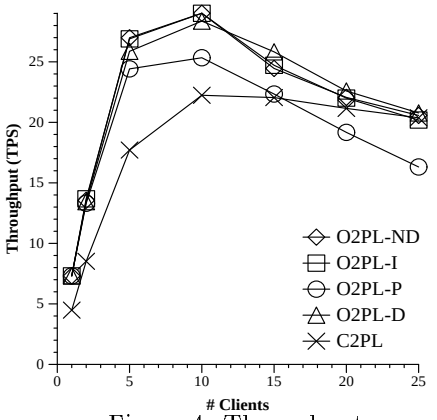

Figure 4: Throughput
(HOTCOLD, 25% Cli Buffers, Fast Net)

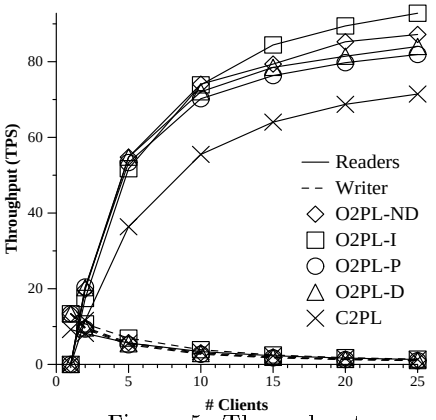

Figure 5: Throughput
(FEED, 5% Cli Buffers, Slow Net)

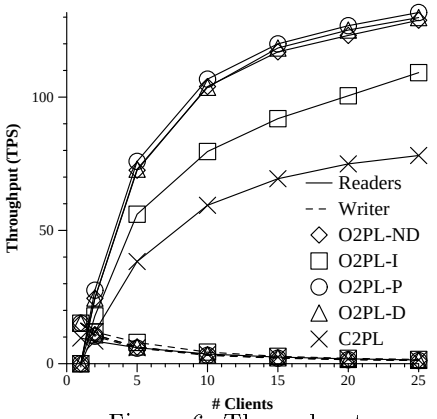

Figure 6: Throughput
(FEED, 25% Cli Buffers, Slow Net)

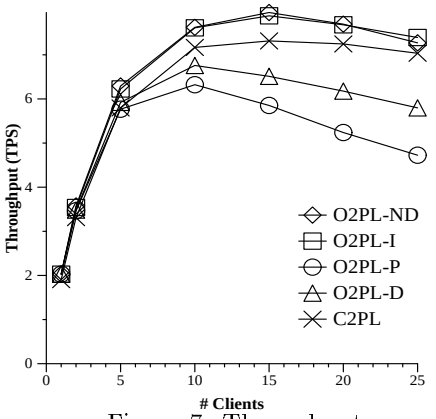

Figure 7: Throughput
(UNIFORM, 25% Cli Buffers, Slow Net)

the heuristic used by the original O2PL-D algorithm. O2PL-ND performs as well as O2PL-I, the better of the static O2PL algorithms in this case, while O2PL-D tracks the lower performance of O2PL-P. All of the algorithms eventually become disk-bound in this case. In this experiment, all of the O2PL algorithms outperform the C2PL algorithm prior to reaching the disk bottleneck. This is due to the additional latency incurred by C2PL's heavy reliance on messages. The similarity of O2PL-D's performance to that of O2PL-P in this case is due to the fact that O2PL-D will propagate a page once to a site before it detects that it should have invalidated the page. Since the client buffer pool is relatively small in this experiment, O2PL-P performs only slightly more propagations than O2PL-D, as an unused page is likely to be pushed out of the buffer pool by the LRU page replacement algorithm. About 80% of the propagations go unused for both O2PL-P and O2PL-D (that is, these propagated pages are not accessed before they are forced out of the buffer pool or another consistency action arrives for the page).

The lower performance of O2PL-P and O2PL-D, as compared to O2PL-I and O2PL-ND, is due to both messages and disk I/O. The additional messages are due to the propagations just described. As was seen in [Carey, 1991a], the additional disk I/O is caused by slightly lower server and client buffer hit rates. These lower hit rates result from the fact that with propagation of updates, pages are removed from the client buffers only when they are aged out. Aged-out pages are not likely to be found in the server buffer pool if they are needed (thus lowering the server buffer hit rate), and propagated pages take up valuable space in the small client buffer pool if they are not needed (thus lowering the client hit rate).

The O2PL-ND algorithm performs similarly to O2PL-I mainly because the majority of its consistency operations are invalidates. This is because the invalidation window is small relative to the database size. As the invalidate window size is increased O2PL-ND becomes more like the original O2PL-D algorithm. Sensitivity tests showed that while the number of propagations per committed transaction increases with the window size, it remains quite small in the range of window sizes tested (10 to 100) and the number of useless propagations grows slowly and smoothly with the invalidate window size in that range. Therefore, there is reasonable room for error in choosing the window size as long as it is kept well below the size of the database.

Figure 3 shows the throughput results for the HOTCOLD workload and slow network when the client buffer size is increased to 25% of the database size (312 pages). This case shows the effect of the updated system parameter settings used in this study. In [Carey, 1991a], which used slower CPUs and a faster network, O2PL-D achieved performance much closer to that of O2PL-I for this case. The earlier parameter settings were based on the intuition that the network itself would not ever be a bottleneck. However, the continuing disparity in the performance increases of CPUs and installed networks has increased the likelihood of a network bottleneck arising. The effect of these technology trends is an increase in the relative cost of useless propagations and as a result, the relative performance of O2PL-D suffers. These trends also exacerbate the performance problems incurred by O2PL-P. In this case, O2PL-P and O2PL-D become network-bound, while O2PL-I and O2PL-ND approach a disk bottleneck at 25 clients. In terms of the new heuristic, the results in this case are similar to those for the 5% client buffer case; O2PL-ND closely matches the performance of O2PL-I because the vast majority of its consistency

operations are invalidations. Furthermore, nearly all of the invalidations are the result of pages not being in the invalidate window (condition 3), so the invalidate window is effective in avoiding even the single mistake that is made by O2PL-D.

We now turn to the throughput results for the HOTCOLD workload using the fast network. When the smaller client buffer pool size is used (not shown), the results are similar to those of the slow network case (Figure 2) in that O2PL-I and O2PL-ND have similar performance, while O2PL-P and O2PL-D have similar but lower performance. However, due to the fast network, the performance differences among the O2PL algorithms are primarily driven by disk I/O requirements. When the larger client buffer size is used in conjunction with the fast network (shown in Figure 4), the original dynamic algorithm performs significantly better than O2PL-P because it performs many fewer propagations than O2PL-P. O2PL-P becomes CPU-bound at the server due to message processing overhead, while all other algorithms eventually become disk-bound. At 15 clients and beyond, O2PL-D matches and even slightly exceeds the performance of O2PL-I and O2PL-ND. This is due to the fact that O2PL-D has a better client hit rate than these other algorithms, and thus requires fewer disk I/Os per transaction. Its improved hit rate is due to the fact that O2PL-D performs more than twice as many useful propagations as O2PL-ND. Of course, O2PL-D also performs many more wasted propagations than O2PL-ND, but the overhead of these propagations is not high enough here to cause O2PL-D to become network or server CPU-bound and the large buffer-pool size ensures that hot range pages will remain in the client buffer pool despite the wasted propagations.

4.1.2 Experiment 2: The FEED Workload [3]

In the FEED workload, client 1 reads and writes data pages, while all other clients only read data pages, and thus, there is a flow of information from client 1 to the others. In contrast to the HOTCOLD workload, where propagation is generally a bad idea, the FEED workload was shown in [Carey, 1991a] to benefit from propagation. This result also holds in general in this study, however, an exception arises in the case with small client buffer sizes and the slow network (Figure 5). In this case, O2PL-I has the highest reader throughput at 15 clients and beyond, while O2PL-P has the lowest reader throughput of the O2PL algorithms. The dynamic algorithms fall roughly between the two static ones, with O2PL-ND having a slight advantage. This ordering is due to the combination of the slow network, which makes propagations expensive, and the small buffer pool, which causes propagations to be wasted. For the O2PL algorithms that do propagation, 53% to 60% of the propagations actually prove to be useful. Moreover, due to the small client buffer pools, many of these propagations are used only once before the page is thrown out. A propagation costs a page-sized message. On the other hand, a propagation that is used exactly once saves only a smaller control message (the page request message that would be required to obtain the page from the server). Thus, if many "useful" propagations are used only once before being thrown out of the buffer pool or re-propagated to, the net effect of a small majority of useful propagations can be a reduction in the number of messages but an increase in the demand for network bandwidth. This is the effect seen in Figure 5.

[3] Due to space limitations, we report results only for the slow network case here.

The FEED workload with small client buffer pools was the only situation examined where the invalidate window size of O2PL-ND made a noticeable performance difference within the range of 10 to 100 pages. In fact, differences are only noticed at window sizes of 50 and smaller, as a window size of 50 pages allows all pages that are updated by the FEED workload to fit in the window. The sensitivity in this case results from the combination of the small client buffer pools and the small number of pages that are updated in the FEED workload. Because of the small buffer pools, hot region pages are often removed by being aged out rather than invalidated. If the window size is less than 50, an aged-out page may not have an entry in the window when it is re-referenced because its entry was pushed out by invalidations of other pages. Therefore, a hot page that is marked as recently invalidated, but then aged out, can lose its marking on a subsequent re-reference. A window size of 50 avoids this problem for the FEED workload, since only 50 pages are ever updated. Therefore a small decrease in the window size raises the possibility of losing the markings of aged-out pages. While the algorithm is sensitive to the window size in this case, its performance is bracketed by O2PL-I (i.e., a window size of 0) and O2PL-D (i.e., a window size of 50). In this test, at 10 clients and beyond, O2PL-ND (window size = 20) chooses to invalidate remote copies about 40% of the time, while O2PL-D chooses to invalidate only about 15% of the time.

Figure 6 shows the throughput of the FEED workload with the slow network when the client buffer pool size is increased to 25% of the database size. As in the previous case, all of the algorithms approach a network bottleneck; in this case however, O2PL-P significantly outperforms O2PL-I. The larger buffer pools allow all of the clients to keep their 50 hot range pages in memory, so many fewer of O2PL-P's propagations are wasted due to pages being forced out of the buffer pool. As a result, O2PL-I sends more page requests to the server than O2PL-P, which results in increased path length for transactions under O2PL-I. This additional path length includes additional messages (O2PL-I sends about 6 messages per transaction versus about 3 per transaction for O2PL-P), and additional lock requests at the server. The two dynamic algorithms have nearly identical performance in this case; they are slightly worse than O2PL-P but much better than O2PL-I. O2PL-D and O2PL-ND perform similarly because O2PL-ND's additional condition for propagation almost always holds here. Due to the large client buffer size, hot range pages are aged out of the buffer pool very infrequently — they are removed mainly as the result of invalidations. Invalidated hot pages are likely to be re-referenced quickly, and if so, they will still be in the invalidate window when the re-reference occurs. Therefore, in this case, a small invalidate window is sufficient to detect invalidated hot pages, so O2PL-ND is fairly insensitive to the invalidate window size.

4.1.3 Experiment 3: The UNIFORM Workload

So far we have investigated one workload where invalidation is advantageous and one in which propagation performs well. We now briefly examine the UNIFORM workload, in which caching is not expected to provide much of a performance benefit. In [Carey, 1991a], there was very little performance difference among the algorithms when the UNIFORM workload was run with the 5% client buffer pool size. Here, this holds only for the fast network case. In the slow network case with the small client buffer pools (not shown), O2PL-P and O2PL-D perform noticeably below the level of the other O2PL algorithms due to the message costs for wasted

propagations. More significant differences arise when the client buffer size is increased to 25%. As shown in Figure 7 , when the slow network is used, O2PL-I and O2PL-ND perform similarly, and both are much better than O2PL-P and O2PL-D. In this workload, propagations do not generally turn out to be useful (e.g., fewer than 15% of O2PL-P's propagations are useful in this case) because the workload's lack of locality means that pages are likely to be aged out of the buffer pool or updated elsewhere before they are accessed again at a site. Therefore, doing fewer propagations results in sending fewer messages and better performance in this case. O2PL-P and O2PL-D approach a network bottleneck at 25 clients, while the other algorithms become disk-bound. When the fast network is used (not shown), the network becomes less of a factor as the disk becomes the main bottleneck. In this case, O2PL-I and O2PL-ND have a slight performance advantage due to better server and client buffer hit rates that result from their use of invalidations.

4.1.4 Summary

The experiments presented in Sections 4.1.1 thru 4.1.3 show that the new heuristic for the dynamic algorithm performs as well as the static O2PL-I algorithm in cases where invalidation is the correct approach, which O2PL-D was unable to do. In addition, it retains the performance advantages of the O2PL-D heuristic in cases where propagation is advantageous. Though the new heuristic never significantly outperformed the better of the static algorithms in any of the cases tested, it was shown to be a reasonable choice even in situations with static locality.

4.2 CALLBACK LOCKING

In this section, we study the performance of the two Callback Locking algorithms: CB-Read and CB-All. For comparison purposes, we also show the performance of O2PL-ND and C2PL. Experiments were run with client buffer pool sizes of 5% and 25% of the database size. Results are shown only for the 25% case since the important aspects of the performance of the CB algorithms are slightly more pronounced (but not qualitatively different) in the large client buffer pool case.

4.2.1 Experiment 4: The HOTCOLD Workload

Figure 8 shows the results of the HOTCOLD workload using the slow network and a client buffer pool size of 25%. The C2PL algorithm has the lowest throughput due to the combination of high message requirements (shown in Figure 9) and its higher disk requirements (compared the others) due to lower buffer hit rates. In general, the CB algorithms both perform at a somewhat lower level than O2PL-ND, which has the highest throughput overall. These results are driven primarily by the message requirements of the algorithms, as the disk requirements of the two CB algorithms and the O2PL-ND algorithm are nearly identical. This is because the CB algorithms are invalidation-based, and O2PL-ND chooses to use invalidation for over 90% of its consistency operations. As shown in Figure 9, the CB algorithms send significantly more messages per commit than O2PL-ND. The difference between CB-Read and O2PL-ND is due to the fact that CB-Read performs consistency operations on a per-page basis, while O2PL-ND performs consistency operations only at the end of the execution phase. O2PL-ND saves messages by sending fewer requests to the server for consistency actions (one per transaction versus as many as one per write for CB-Read) and, to a lesser extent, by grouping multiple consistency requests for

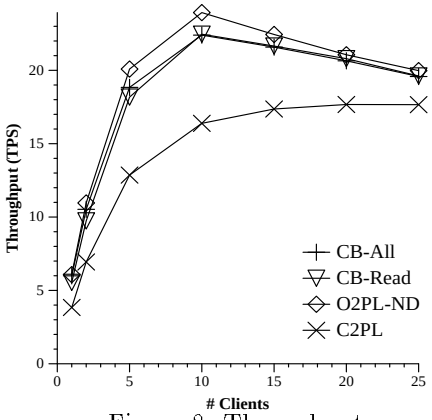

Figure 8: Throughput
(HOTCOLD, 25% Cli Buffers, Slow Net)

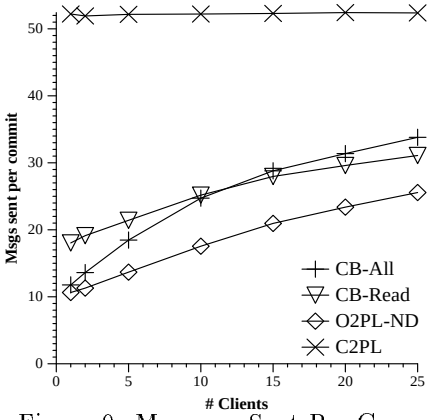

Figure 9: Messages Sent Per Commit
(HOTCOLD, 25% Cli Buffers, Slow Net)

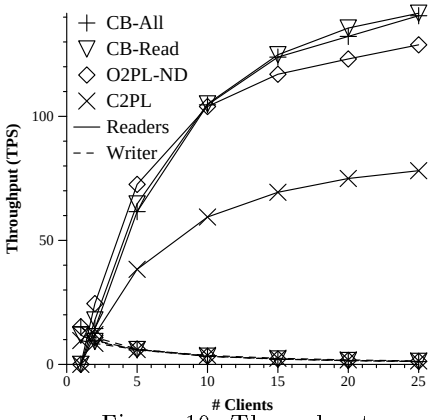

Figure 10: Throughput
(FEED, 25% Cli Buffers, Slow Net)

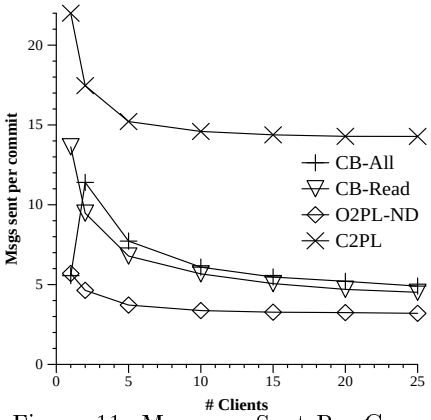

Figure 11: Messages Sent Per Commit
(FEED, 25% Cli Buffers)

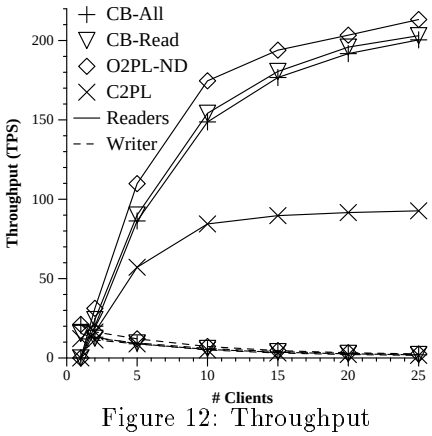

Figure 12: Throughput
(FEED, 25% Cli Buffers, Fast Net)

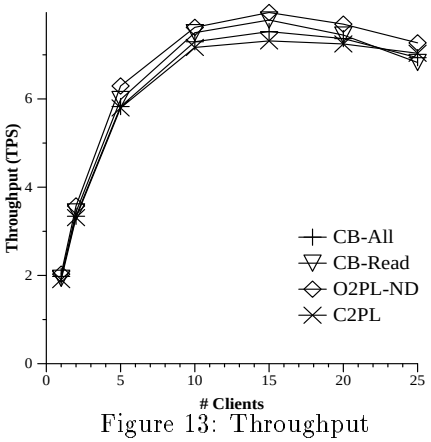

Figure 13: Throughput
(UNIFORM, 25% Cli Buffers, Slow Net)

the same client in a single message. In contrast to CB-Read, CB-All is allowed to cache write locks; this is the reason for its sending fewer messages than CB-Read when small numbers of clients are present. However, beyond 10 clients the caching of write locks results in a net increase in messages. For the CB algorithms, the advantage of caching a write lock is that a request for a lock on a site where it is already cached saves one round trip message with the server. However, the penalty for caching a write lock that conflicts with a read request at a different site is also a round trip message with the server. In the HOTCOLD workload, caching write locks becomes a losing proposition when the number of clients is sufficient to make it more likely that a page will be read at some remote site before it is re-written at a site with a cached write lock. When the fast network is used (not shown), all of the algorithms eventually become disk-bound so the messaging effects discussed in the slow network case eventually have no impact on the throughput. As a result, CB-Read, CB-All and O2PL-ND all have similar performance.

4.2.2 Experiment 5: The FEED Workload

The reader and writer throughput results for the FEED workload with the slow network are shown in Figure 10. O2PL-ND has the best reader throughput prior to 10 clients, while CB-Read and CB-All have the best reader throughput beyond 10 clients. The writer throughput is similar for all of the algorithms at 5 clients and beyond. O2PL-ND and CB-All have better writer throughput when there are 0 or 1 reader sites. Once again, the throughput results are driven by the message requirements of the algorithms. Figure 11 shows the messages sent per commit averaged over the writer and all of the readers for each of the algorithms.[4] O2PL-ND's higher initial reader throughput is due to its lower message requirements, which result from its reduced consistency message requirements (as described previously) and a better client buffer hit rate due to propagations. All of the algorithms except for C2PL approach a network bottleneck at 15 clients and beyond. As the network becomes saturated, the cost of wasted propagations performed by O2PL-ND cause its reader performance to suffer compared to the callback algorithms. In terms of the callback algorithms, the writer site in CB-Read has to request every write lock from the server, while in the CB-All algorithm, the writer has to request write locks that have been called back (virtually all write locks at five clients and beyond). CB-All also requires extra messages for readers to callback the write locks that are cached at the writer site. As a result, CB-All's writer site receives and sends up to 6 times more messages per commit than CB-Read's writer site in this case. The callback requests for write locks are the cause of the difference in the message requirements for the CB algorithms seen in Figure 11. Furthermore, there is little or no benefit to caching write locks here, as the updated pages are in the hot region of all of the readers and are thus likely to be read at reader clients. The C2PL algorithm has the highest message requirements for reader sites, as such sites must send a message to the server for each page accessed. Many of these messages are small, so their main impact on C2PL is an increase in server and client CPU requirements.

The throughput results for the FEED workload with the fast network are shown in Figure 12. Again, in this case, all of the algorithms except for C2PL approach

[4]Since client number 1 is the writer site, the data points for one client in Figure 11 show the messages per writer transaction in the absence of conflicting readers.

(but do not quite reach) a disk bottleneck at 25 clients. C2PL eventually becomes server CPU-bound due to lock requests that are sent to the server. The propagations performed by O2PL-ND give it a better buffer hit rate at the reader clients. However, this translates to only a slight reduction in disk reads compared to the callback algorithms, as hot pages missed at clients due to invalidations are likely to be in the server's buffer pool. Thus, in the range of clients studied, the relative performance of the algorithms is still largely dictated by the message characteristics as described for the slow network case. With the fast network, the size of the messages is less important. Therefore the relative performance of the algorithms is more in line with the message counts shown in Figure 12.

4.2.3 Experiment 6: The UNIFORM Workload

The results for the UNIFORM workload using the slow network are shown in Figure 13. None of the algorithms hits a resource bottleneck in the range of clients shown. O2PL-ND achieves the highest throughput across the range of client populations, with the callback algorithms performing below O2PL-ND but better than C2PL through most of the range. CB-Read performs slightly better than CB-All because caching write locks is costly due to the lack of locality and the low write probability. The decline in throughput for the three lock caching algorithms is due to their increased message requirements for consistency operations in this low-locality workload. At 15 clients and beyond, all three of the lock caching algorithms send more messages than C2PL, which does not cache locks. While C2PL's message requirements per commit remain constant as clients are added, the lock caching algorithms are forced to send consistency operations (e.g., invalidations or callbacks) to more sites. C2PL's lower performance through most of the range is due to its higher disk requirements resulting from low client and server buffer hit rates. The reason that the algorithms do not quite reach a bottleneck in this case is a higher level of data contention than was seen in the other workloads. When the fast network is used (not shown), the network effects are removed as all algorithms eventually approach a disk bottleneck. Therefore, the lock caching algorithms perform similarly, and slightly better than C2PL due to buffering characteristics.

4.2.4 Experiment 7: The HICON workload

The final workload examined in this paper is the HICON workload. While we do not expect high data contention to be typical for client-server DBMS applications, we use this workload to examine the robustness of the algorithms in the presence of data contention and to gain a better understanding of their different approaches to detecting conflicts. As described in Section 3.2, this workload has a 250 page hot range that is shared by all clients. The write probability for hot range pages is varied from 0% to 50% in order to study different levels of data contention. In this section we briefly describe the HICON results, using the fast network and large client buffer pools. Figure 14 shows the throughput for HICON with a hot write probability of 5%. In the range of 1 to 5 clients, the lock caching algorithms perform similarly and C2PL has the lowest performance. C2PL's lower performance in this range is due to its significantly higher message requirements here. These results are latency-based, as no bottlenecks develop in this range — in fact, no resource bottlenecks are reached by any of the algorithms in this experiment. At 10 clients and beyond, the effects of increased data contention become apparent; O2PL-ND's performance suffers and it

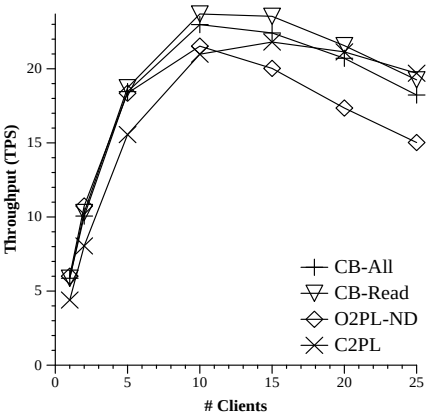

Figure 14: Throughput
(5% HICON, 25% Cli Buffers, Fast Net)

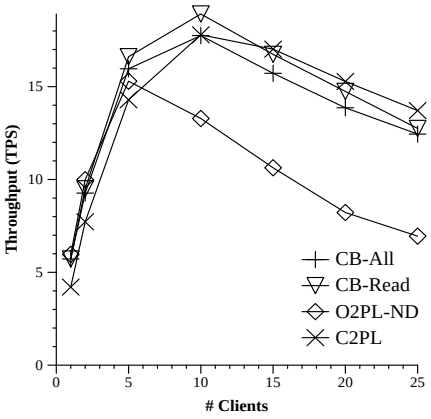

Figure 15: Throughput
(10% HICON, 25% Cli Buffers, Fast Net)

has the lowest utilization of all three major system resources (disk, server CPU, and network). O2PL-ND has a somewhat higher level of blocking for concurrency control than the other algorithms (e.g., at 15 clients, approximately 42% of its transactions are blocked at any given time versus approximately 37% for the callback algorithms and 35% for C2PL). O2PL-ND has a higher blocking level because it detects global deadlocks using periodic detection, while the other algorithms detect deadlocks immediately at the server. All of the algorithms suffer from increased blocking as clients are added. In addition, the lock caching algorithms suffer from increasing message costs due to consistency messages as clients are added. These two factors account for the thrashing behavior seen in Figure 14. Again, in this workload, the caching of write locks causes CB-All to send more messages than CB-Read. C2PL performs best at 25 clients because at that point it sends the fewest messages. C2PL's message requirements remain fairly constant as clients are added to the system, while the other algorithms all incur an increase in messages for consistency operations as clients are added. The slight downturn in performance seen for C2PL beyond 15 clients is due to data contention.

Figure 15 shows the throughput of the algorithms when the hot write probability is increased to 10%. Here, the trends seen in the 5% write probability case are even more pronounced. C2PL becomes the highest performing algorithm at 15 clients and beyond, and O2PL-ND performs at a much lower level than the other algorithms. C2PL sends fewer messages per transaction than the other algorithms at fifteen clients and beyond in this case. An increase in data contention causes all of the algorithms to exhibit thrashing behavior beyond 10 clients. In this case, the thrashing is due not only to additional blocking, but also to a significant number of aborts. For example, at 20 clients, O2PL-ND performs nearly 0.6 aborts per committed transaction, while the other algorithms perform about 0.3 aborts per commit. In all of the HICON experiments, O2PL-ND was found to have a significantly higher abort rate than the other three algorithms. This is due to the fact that it resolves write-write conflicts using aborts, whereas the other three algorithms resolve them using blocking. The trends seen in these two cases become more pronounced as the hot write probability is increased beyond the two cases shown here. An interesting effect that appears with higher write probabilities (e.g.,

HICON with write probabilities of 25% and 50%) is that in some cases, O2PL-ND actually has fewer blocked transactions than the other algorithms. This effect arises because of its higher abort rate, which at high contention levels acts as a throttle on the blocking level. However, the net result is that O2PL-ND performs worse relative to the other algorithms as the hot write probability is increased.

4.2.5 Summary

The results described in Sections 4.2.1-4.2.4 show that the CB algorithms have similar (but often slightly lower) performance to O2PL-ND. The CB algorithms typically have higher message requirements than O2PL-ND because they perform consistency maintenance on a per-page basis. Therefore, they perform below the level of O2PL-ND in situations where network usage plays a large factor in determining performance. However, in situations where disk I/O is the dominant factor, they perform comparably to O2PL-ND. The caching of write locks (by CB-All) decreased message traffic only in cases with few clients or limited data contention, and resulted in additional message costs in most other cases. The CB algorithms performed much better than C2PL under most workloads, while retaining the lower (compared to O2PL-ND) abort rate of that algorithm. This is because the CB algorithms cache locks across transactions without using optimistic techniques. The combination of pessimism and the ability to perform deadlock detection locally at the server allow the CB algorithms to be much more robust in the presence of data contention than O2PL-ND. C2PL had the best performance under very high data contention because it has a low abort rate and fast deadlock detection (similar to CB) together with message requirements that remain constant as client sites are added to the system. However, in all other cases, C2PL performed below the level of the CB algorithms and the better of the O2PL algorithms.

5 RELATED WORK [5]

5.1 DYNAMIC ALGORITHMS

We are unaware of any other work investigating dynamic algorithms for choosing between propagation and invalidation to maintain cache consistency in a client-server DBMS environment. Similar problems have been addressed in work on Non-Uniform Memory Access (NUMA) architectures, however. In such systems, an access to a data object that is located in a remote memory can result in migrating the data object to the requesting site (invalidation), replicating the object (propagation), or simply accessing the object remotely using the hardware support of a shared-memory multiprocessor. These issues are addressed in Munin [Carter, 1991] by allowing the programmer to annotate data declarations with a designation of the sharing properties of each variable. These annotations serve as hints to Munin in deciding among propagation and invalidation, making replication decisions, etc. DUnX [LaRowe, 1991] uses a parameterized policy which chooses dynamically among page replacement options based on reference histories. The parameters allow a user to adjust the level of dynamism of the algorithms. [LaRowe, 1991] demonstrated that it was possible to find a set of default parameter settings that provided good performance over a range of NUMA workloads.

[5] In this section we discuss work related to the specific issues that were addressed in this study. For work related to client-server caching in general see [Carey, 1991a].

5.2 CALLBACK LOCKING

A Callback Locking algorithm for client-server database systems was studied in [Wang, 1991]. That study compared the callback algorithm with a caching 2PL algorithm (similar to C2PL) and two variants of a "no-wait" locking algorithm that allowed clients to access cached objects before receiving a lock response from the server. The results of that study showed that if network delay was taken into account, Callback Locking was the best among the algorithms studied when locality was high or data contention was low, with caching 2PL being better in high conflict situations. These results mostly agree with those presented in Section 4.2, but there are several differences between the two studies. First, the workload model used in [Wang, 1991] provided a notion of locality that was more dynamic than that of this study but was not as flexible in terms of the nature of data sharing among clients. [Wang, 1991] did not study the implications of caching write locks, and did not examine algorithms that have lower message requirements such as O2PL-ND. Also, as described in Section 2.3, the Callback Locking algorithm in [Wang, 1991] differed from CB-Read in its method of detecting deadlocks.

An area that is closely related to client-server caching is work on using Distributed Shared Virtual Memory (DSM) to support database systems [Bell, 1990]. DSM is a technique that implements the abstraction of a system-wide single-level store in a distributed system [Li, 1989]. Three algorithms for maintaining cache consistency were studied in [Bell, 1990], one of which was a 2PL variant that used callbacks. The callback algorithm was compared to several broadcast-based algorithms using a simulation model that had an inexpensive broadcast facility. Given this facility, the callback-style algorithm typically had lower performance than one of the broadcast algorithms. Several related algorithms for the shared-disk environment have been investigated in [Dan, 1992]. These algorithms include shared-disk equivalents of the C2PL, CB-Read and CB-All algorithms. An algorithm that was similar to CB-Read (i.e., it retained only read locks) was shown to perform relatively well in that study.

As mentioned earlier, callback algorithms were initially developed for use in distributed file systems. The Andrew File System [Howard, 1988] uses a callback scheme to inform sites of pending modifications to files that they have cached. This scheme does not guarantee consistent updates, however. Files that must be kept consistent, such as directories, are handled by simply not allowing them to be updated at cached sites. Sprite [Nelson, 1988] provides consistent updates, but it does so by disallowing caching for files that are open for write access. This is done using a callback mechanism that informs sites that a file is no longer cachable.

6 CONCLUSIONS

In this paper, we have extended the results of the earlier client-server caching study of [Carey, 1991a]. A new heuristic for the dynamic O2PL algorithm was shown to perform as well as a static invalidation-based O2PL algorithm in cases where invalidation is the correct approach — which the heuristic of [Carey, 1991a] was unable to do. In addition, it retains the performance advantages of the earlier heuristic in cases where propagation is advantageous. The heuristic uses a fixed window size parameter, but it was found to be fairly insensitive to the exact size of

the window as long as the window size was kept small in proportion to the database size. The advantages of the new heuristic were more significant than were seen with the parameters of [Carey, 1991a], as the greater disparity between CPU speeds and network bandwidth here (when the slow network was used) increased the negative effects of bad propagations compared to what was seen in that study. Two variants of Callback Locking were described and studied: CB-Read, which caches only read locks, and CB-All, which caches both read and write locks. CB-Read was found to perform as well as or slightly better than CB-All in many situations, because the caching of write locks caused a net increase in messages except in cases with small client populations or minimal data contention. Both CB algorithms were seen to have slightly lower performance than O2PL-ND (which typically sent fewer messages) in situations where network usage plays a large factor in determining performance but their performance was similar to that of O2PL-ND in cases where disk I/O was the dominant factor. However, the CB algorithms were seen to have a lower abort rate than O2PL-ND, and were much more robust than O2PL-ND in the presence of data contention. A low abort rate is particularly important in a workstation-based environment, as the abort of a complex transaction is likely to be less acceptable to users than the abort of a simple transaction (e.g., in a Transaction Processing environment). Finally, C2PL, which does not cache locks, was found to be the best algorithm for workloads with very high data contention, but performed below the level of the CB algorithms and O2PL-ND in all other cases.

The results of this study have led us to choose CB-Read for implementation in the EXODUS system. This choice is due to its combination of good performance, low abort rate, and relative ease of implementation. This implementation effort is currently under way. In addition, a number of issues remain to be addressed in the area of cache consistency algorithms. In particular, mixed workloads and workloads with dynamic properties should be studied to better demonstrate the effectiveness of the adaptive algorithms. In addition, it should be possible to devise adaptive versions of the callback algorithms that perform better in situations where propagation is appropriate. It should also be possible to devise algorithms that are adaptive in terms of lock caching as well as data caching, in order to handle high contention "hot spot" data. Finally, we are investigating ways of more fully exploiting the memory and processing power of the workstations in a client-server database system. A first step in this direction is described in [Franklin, 1992a].

Acknowledgements

We thank Miron Livny and Beau Shekita for providing insight into the performance of client-server systems. We also thank Dave Maier and the database reading group at OGI for providing helpful comments on an earlier version of this paper. This work was partially supported by DARPA under contract DAAB07-92-C-Q508, by the National Science Foundation under grant IRI-8657323, and by a research grant from IBM.

References

M. Bellew, M. Hsu, and V. Tam. (1990) Update Propagation in Distributed Memory Hierarchy. In*Proc. 6th IEEE Data Eng. Conf.*, Los Angeles.

M. Carey, M. Franklin, M. Livny, and E. Shekita. (1991a) Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proc. ACM SIGMOD Conf.*, Denver.

M. Carey, and M. Livny. (1991a) Conflict Detection Tradeoffs for Replicated Data. In *ACM Transactions on Database Sys.*, 16(4).

J. Carter, J. Bennett, and W. Zwaenepoel. (1991) Implementation and Performance of Munin. In *Proc. 13th ACM Symp. on Op. Sys. Principles*, Pacific Grove, CA.

A. Dan and P. Yu. (1992) Performance Analysis of Coherency Control Policies through Lock Retention. In *Proc. ACM SIGMOD Conf.*, San Diego.

O. Deux *et al.* (1991) The O2 System. *Comm. of the ACM*, 34(10).

D. DeWitt *et al.* (1990) A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems, In *Proc. 16th VLDB Conf.*, Brisbane, Australia.

EXODUS Project. (1991) *EXODUS Storage Manager Architectural Overview.* Univ. of Wisconsin-Madison, November.

M. Franklin, M. Carey, and M. Livny. (1992a) Global Memory Management in Client-Server DBMS Architectures. In *Proc. 18th VLDB Conf.*, Vancouver, B.C., Canada.

M. Franklin, M. Zwilling, C. Tan, M. Carey, and D. DeWitt. (1992b) Crash Recovery in Client-Server EXODUS. In *Proc. ACM SIGMOD Conf.*, San Diego.

M. Hornick and S. Zdonik. (1987) A Shared, Segmented Memory System for an Object-Oriented Database. *ACM Trans. on Office Info. Sys.* 5(1).

J. Howard. *et al.* (1988) Scale and Performance in a Distributed File System. *ACM Trans. on Computer Sys.*, 6(1).

C. Lamb, G. Landis, J. Orenstein, and D. Weinreb. (1991) The ObjectStore Database System, *Comm. of the ACM*, 34(10).

P. LaRowe, C. Ellis, and L. Kaplan. (1991) The Robustness of NUMA Memory Management. In *Proc. 13th ACM Symp. on Op. Sys. Principles*, Pacific Grove, CA.

K. Li and P. Hudak. (1989) Memory Coherence in Shared Virtual Memory Systems. *ACM Trans. on Computer Sys.*, 7(4).

M. Livny. (1988) *DeNet User's Guide*, Version 1.0, Comp. Sci. Dept., Univ. of Wisconsin-Madison.

M. Nelson, B. Welch, and J. Ousterhout. (1988) Caching in the Sprite Network File System. *ACM Trans. on Computer Sys.* 6(1).

M. Stonebraker. (1979) Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES. *IEEE Trans. Software Eng.* SE-5(3).

Y. Wang and L. Rowe. (1991) Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In *Proc. ACM SIGMOD Conf.*, Denver.

W. Wilkinson, and M. Neimat. (1990) Maintaining Consistency of Client Cached Data. In *Proc. 16th VLDB Conf.*, Brisbane, Australia.