Interoperable Server-based Cache Consistency Algorithm

Peter Bodorik*, Dawn Jutla**, and Yueping Lu***

***Abstract* — Numerous caching algorithms have been investigated for the client-server object databases management systems. The algorithms not only ensure cache consistency by preventing applications' access to stale data, but they also support transactional properties by ensuring consistent access, to the cached data at the numerous clients, thereby satisfying the DB integrity constraints. Caching algorithms have been classified in a number of ways – one classification is into avoidance and detection categories, depending on whether access to the stale data is avoided, usually by locking, or permitted and then any conflict detected at commit time. Detection-based algorithms have better performance but can lead to high abort rate that is unacceptable for interactive applications. It is for this reason that avoidance-based algorithms are usually adopted in practice. This paper describes a server-based interoperable transactional caching algorithm that concurrently supports the leading avoidance-based (*adaptive callback locking (ACBL)*) and detection -based (*adaptive optimistic concurrency control(AOCC)*) algorithms. At a client either the avoidance or the detection caching algorithm is used without any changes. It is the server-side caching algorithm that concurrently supports both avoidance and detection client-side caching.**

***Index Terms*— Caching, Cache Coherence, Transactional cache, Object Oriented Database, Integrated Coherence and Concurrency Control Algorithms, Client-server architecture, Interoperability, Cache server**

I. Introduction

Caching is an important technique used to improve performance of applications that access DB systems. Client-server architectures utilize either *query shipping* or *data shipping.* In relational DBMS that use the client-server architecture, query shipping is utilized. The query is shipped to the server that executes it and sends the result to the client where it is cached. Object DBMS or object-relational DBMS, on the other hand, utilize data shipping, in which the client requests data/objects while queries are executed by the client. The server is thus off-loaded by executing queries on the clients and, furthermore, the locality of reference, exhibited by applications that navigate through complex data structures, is exploited.

Applications that access DBs require transactional support. For efficient caching in support of transactions, caching methods have been developed to also support transactional properties. Thus transactional caches integrate the caching and concurrency control functions and perform these far more efficiently then if these functions were performed independently [Franklin 1997, Bodorik 1998].

A taxonomy of transactional caching methods has been proposed in [Franklin, 1997]. The first classification is based on one of two methods that deal with potential access to invalid/stale data. Stale data is such that it has been updated in the DB by some other application. There are two basic methods: detection and avoidance. A detection-based algorithm lets a transaction access locally cached data even though it may be stale. It checks whether any of the data accessed by the transaction is stale later, usually during the transaction's commit. A transaction that has read data modified by other transactions is aborted. An avoidance-based algorithm prevents a transaction to see stale data in the first place. Detection-based algorithms are further classified on when validation is performed (synchronous, asynchronous, or deferred until commit), change notification hints (none or after-commit), and remote update action (invalidation, propagation, or dynamic). Avoidance-based algorithms are further categorized by write intention declaration (synchronous, asynchronous, or deferred to commit), write permission duration (end of transaction or until revoked/dropped), remote conflict priority (wait or preempt), and remote update duration. Although the taxonomy applies to caching that uses data shipping, we claim that with some modifications the above taxonomy can also be applied to caching methods that use query shipping. In [Voruganti 1999], classification of algorithms concentrates on the granularity of data, either pages or objects (in a page), for the purposes of data transfer, cache consistency, buffer management, recovery, and pointer swizzling. *Adaptive callback locking (ACBL)* has been considered to be the leading avoidance-based caching algorithm, while *adaptive optimistic concurrency control (AOCC)* has been considered as the leading detection-based algorithm.

Studies that have been performed to investigate the performance of the algorithms generally use simulation with the workload being based on the OO7 benchmark [Carey 1994]. Simulations model clients that execute one transaction at a time on systems with limited buffer space and given CPU speed (instructions per second). Transaction requests generate load on the CPU (number of instructions) and also requests for data (pages, objects). Transaction's access to data, whether it is read/write, and whether it is locally cached, are governed by probability distributions. If data is not available locally, it is fetched from the server, which generates load on

This work was supported in part by a grant from the National Science and Engineering Research Council of Canada.

*Peter Bodorik is a professor in Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada, Peter.Bodorik@cs.dal.ca.

**Dawn Jutla is an associate professor in Faculty of Commerce, Saint Mary's University, Canada , Dawn.Jutla@stmarys.ca.

***Yueping Lu is a graduate student in Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada.

the CPU and also on the network. Network cost to transfer a message has fixed (set-up) and variable (per byte) components. The server has a CPU of a given speed, limited buffer space, and a number of disks. In addition to caching, simulation may also take into account recovery requirements by modeling update activities.

There have been a number of performance studies that have compared the various algorithms for object DBs under varying scenarios [Carey 1991, Wang 1991, Chang 1994, Carey 1994, Adya 1995, Chang 1997, Franklin 1997, Ozsu 1998, Voruganti 1999]. It has been concluded that under expected workloads invalidation performs better than update propagation and thus invalidation has been adopted by most caching algorithms. Comparison of the avoidance-based algorithms and detection-based algorithms generally concluded that detection-based algorithms, such as AOCC, outperform avoidance-based algorithms, such as ACBL, even in situation where AOCC encounters a high abort-rate. In [Ozsu 1998] it has been observed that this is due to the low cost of abort and transaction's re-execution. It is assumed that the clients have sufficient memory to avoid eviction of data due to shortage of buffer space. As a consequence, if a transaction is aborted, most pages in the transaction's working set are already present in the cache when the transaction is re-executed.

An important observation made in [Ozsu 1998] is that in spite of the fact that studies show that avoidance-based algorithms outperform synchronous detection ones, most commercial client caches use ACBL or its variants because the cache is used by interactive applications for which a high abort rate is unacceptable. Restated, the problem is that although various applications may benefit from the good performance of the AOCC caching algorithms, these applications are forced to use ACBL that is required for (interactive) applications that cannot tolerate high abort rate.

There are two approaches to this problem. One is to continue towards an efficient algorithm that is mostly avoidance based but in performance is close to the detection-based AOCC. Indeed, this approach has been taken in [Voruganti 1999] by proposing a new algorithm, called *asynchronous-based cache consistency algorithm (AACC)*, which is claimed to have a low abort rate and a good performance. Both the server and the clients manage locks at the page and also object levels. Each page can be locked in a private read mode, shared read mode, and write mode. Different transactions can write to different objects on the same page. After a transaction's commit, pages that were written by the committed transaction are retained in the client cache in the private-read mode (as private-read locked). If a page is read at more than one location then it is locked in a shared-read mode. If a transaction needs to modify a page it must first be locked by the client cache in the write mode. Asynchronous messages that are piggy-backed are used for changes in the lock-modes. The server has to perform deadlock detection when conflicting operations are performed by the transactions in conflicting modes on the same object and also sends callback messages if there is no deadlock to force a change of state from shared-read to write.

Another approach to this problem is to allow both algorithms to interoperate in the same environment. This is the approach we have adopted. We allow the clients to specify whether they want to use the AOCC or ACBL caching algorithm. Furthermore, the caching operations of the client-side algorithms are not affected – only the server-side is changed. Thus, in the environment in which the ACBL (AOCC) caching algorithm is used and new applications are introduced that should use the AOCC (ACBL) algorithm, we can do so without affecting already existing clients.

The paper is organized as follows. The second section provides further background on the relevant caching algorithms. The third section presents requirements and assumptions. The interoperable server is described in the fourth section. Relevant literature is reviewed in the fifth section. The final section provides summary and conclusions.

II. Background

This section reviews in detail the two, considered to be leading, caching algorithms. One is avoidance-based, ACBL, while the other one is detection-based optimistic algorithm, AOCC. Recall, that the following sections present a server that will support both types of caching clients and thus we shall concentrate on the interaction between the clients and servers.

When a transaction completes, pages that the transaction has accessed are retained and are accessible by subsequent transactions. Furthermore, for simplicity but without loss of generality, it is assumed that each client executes only one transaction at a time. Pages have version identifications that enable the server to determine whether a cached page and the page on the server are the same or whether the server's page is newer and the cached page is thus stale.

A. AOCC

The AOCC is a simple optimistic algorithm that allows a transaction to access locally cached data and defers checking whether the data is stale until the transaction's commit phase. If a page is not in the local cache it is requested from the server that delivers it without delay. Upon the transaction's commit request, the client sends a commit-request message to the server together with the transaction's read and write sets, where the read and write sets are the sets of pages respectively read and written by the transaction. The server determines whether or not any of the pages in the read and write sets of the transaction are stale (have been updated by some transaction since it was cached). If so, the server aborts the transaction. Otherwise the transaction is committed and each page in the transaction's write set is invalidated at clients other than the transaction's home client.

To present the caching method we shall use a *finite state machine (FSM)* to represent the states and transitions for *a page* on a client and another FSM for pages on the server. Inputs include transactions' actions, client cache action (e.g., evict page), and messages passed between the server and the clients. The client-side state diagram for a page as it is affected by the AOCC client-server interaction is shown in

Figure 1 (figures are located after references and prior to the appendix). The state of a page is affected by the operations of the local transaction (operations are the transaction's read and write), management of the transaction and the local cache (commit or abort transaction and evict a page), and interaction with the server (invalidate a page). The server-side state diagram is shown in Figure 2, while the complete definition of the state transitions are shown in tables A-1 and A-2 in Appendix.

Interaction between the client and the server can be summarized as follows:

- The client requests a page.
- The server supplies a requested page to the client without delay.
- The client requests a commit of a transaction while supplying the server with the transaction's read and write sets.
- Upon the request to commit a transaction, the server aborts the transaction if it has read or wrote stale pages. Otherwise, the server commits the transaction and invalidates all cached data pages modified by the committed transaction (of course, with the exception that the modified pages are not invalidated at the client hosting the committed transaction).
- When a client receives a page's invalidation, it checks whether the page has been read by the local transaction and if so, it aborts the transaction.
- If a page is evicted from the client's buffer pool, the client informs the server of this fact.

Note that page invalidations and page eviction messages are piggybacked and not sent by explicit messages. Local abort of a transaction need not be communicated to the server.

B. *ACBL*

ACBL is a synchronous avoidance-based algorithm as it uses lock-escalation messages in a synchronous manner – it sends a request for lock-escalation and waits for a reply before proceeding. As in AOCC, pages are retained across transaction boundaries by a client cache and the cached pages are assumed to be read-locked. Both the server and the clients keep track of read and write locks first at the granularity of pages and then, in case of conflicts on pages, at the level of objects. If conflicts arise, call-backs are used for resolution. If resolution is not possible due to a deadlock, a transaction is aborted. Without loss of generality it is assumed that only one transaction executes at a client at a time. Furthermore, only one client (transaction) is allowed to write to a page. This simplifies discussion and associated coordination of merging updates in comparison to the case when more than one transaction is allowed to write to different objects on the same page. It is possible, however, that one transaction writes to an object on a page while another transaction(s) on a different client reads objects on the cached copy of the same page as long as they are different objects, that is objects not written to which are write-locked.

The client side stage diagram for a page is shown in Figure 3 (located after references and prior to the appendix). The state of a page is affected by the transaction's reads and writes, by the transaction's commit or abort, eviction of a page, and messages received from the server. The server state diagram for a page is shown in Figure 4. The state transition diagrams show only inputs that effect transitions from one state to another state. Transitions from one state to the same state are not shown in figures. However, definitions of all state transitions are given in tables A-3 and A-4 in Appendix.

The page states for the ACBL **client** (Figure 3) are:

- *Not-cached* – the page is not cached on the client.
- *Cached-Not-locked* – the page is cached but no items have been accessed (read/written) and hence no objects are read-locked or write-locked.
- *Cached-read-locked* – the page is cached and some of the objects have been accessed by the local transaction and are thus read-locked.
- *Not-cached-read-locked* – the page is not cached by the client but it is read locked as some of the objects have been read by the local transaction. Consider a page P that is cached and read-locked in client C1. The page P is also cached and read-locked in a client C2 by a transaction T. The transaction T also acquires a write lock on object O located on the page P, action which involves a callback to the client C1. Assume that T gets a write lock on the object O, writes to it, and subsequently commits. As part of the commitment, the cached copies of the page are invalidated at all clients (here client C1) but at the client C2 where the transaction T committed. Thus, the page P is not cached in C1 but it is read-locked at that site as the local transaction had read objects on the page. If there is another read on the page P at C1, the page will be requested from the server.
- *Exclusive* – the page is cached and it (the whole page) is write-locked in an exclusive write mode by the local transaction.
- *Objects-write-locked-local* – the page is cached and some of its objects are write-locked by the local transaction.
- *Objects-write-locked-foreign* – the page is cached and some of its objects are read-locked by the local transaction while some of the objects are write-locked by a foreign transaction (transaction executing on another client).

The page states for the ACBL **server** (Figure 4) are:

- *Not-cached* – the page is not cached at any clients.
- *Cached-read-locked* – the page is cached at one or more clients that have the page share read-locked.
- *Exclusive* – the page is cached and it (the whole page) is locked in an exclusive write-mode at exactly one client.
- *Objects-write-locked* – the page is cached at one or more clients and at one client the transaction also has write locks on some of the objects.

A whole page is write-locked in an exclusive mode only if there is exactly one cached copy. The server assumes that any cached copy is automatically read-locked. Interaction between the server and clients can be summarized as follows:

- Upon read or write by a local transaction to a page that is not cached, the client requests a page and waits for the server's reply; to request a page for a read, message ACBL-pg-req-rd-locked is used, while for a write the message ACBL-pg-req-wr-locked is used.
- When a transaction attempts a write to an object, to which it does not have a write-lock, that is on a page which is not in the state Objects-wr-locked-foreign, the client sends a lock escalation message, ACBL-wr-lock-req, to the server with identification of the page and object to be written. The request is for a write-lock on the whole page and, if that is not possible (because the page is read-locked at some clients), then for a write-lock on the object the transaction is trying to write. The client waits for the server's response. The server sends a call-back request, a ACBL-callb message that contains the page and object IDs, to all other cached copies – the request is to relinquish the read-lock on the specified object. If a client that received the ACBL-callb message has not read the page, and hence the page is in the Cached-NOT-locked state at the client, then the client invalidates the page by changing its state to Not-cached and sends the server a ACBL-callb-reply message together with the indication that the page was purged locally. If the client has already read the page, and hence the page is in the Cached-read-locked state, then the client checks whether the object was read locally. If not, the client changes the state of the page to Objects-wr-locked-foreign and sends a ACBL-callb-reply to the server with indication that the page has been read locally, but not the particular object. If the object has been read locally, the client does not reply to the request until the transaction, which read the object in question, terminates; only then does the client send a ACBL-callb-reply. When all replies to the callback arrive, the server gives the write-lock to the requesting client by sending it a ACBL-wr-lock-grant message.
- If a page is locked exclusively at client C1 and at a client C2 a transaction issues a read to the page (which is not cached), the client C2 sends the server a request for the page by sending it a ACBL-pg-req-rd-locked message together with the page and object IDs. The server then sends a callback for a write-lock to the client C1, which holds the exclusive lock, by sending it a ACBL-callb-wr-lock message that identifies the page and the object. If the object at the client C1 has not been written to, the client relinquishes the exclusive lock to the page by sending the server a ACBL-callb-wr-lock message and changing the page state to Objects-wr-locked-local. The server then informs C2 by sending it a copy of the page using the ACBL-serve-pg-rd-locked message. If the object was written to at the client C1, ACBL-callb-wr-lock reply is not sent. Eventually, the transaction at C1 terminates, and the server then sends the ACBL-wr-lock-grant the client C2.
- When a transaction is attempting a read to an object that is write-locked by a foreign transaction (the page is in the Cached-wr-locked-foreign state), the local transaction cannot proceed and has to wait for the lock release. It piggybacks the request for a read-lock to the server by sending it a ACBL-rd-lock-req message. Furthermore, when the transaction holding the write-lock commits, the server sends to the waiting client/transaction a ACBL-serve-pg-rd-locked message containing: the invalidation of the page in question (and thus information that all write-locks have been released), the new page, and a grant of shared-read lock on the page. If the transaction holding a write-lock on the page is aborted, the server sends the client requesting a ACBL-rd-lock-grant message.
- When a transaction is successfully committed, the server removes all of the transaction's write locks and invalidates pages modified by the transaction at all clients but the one hosting the committed transaction. Also, if there are waiting requests for the page, the pages is served by using the ACBL-pg-req-rd-locked or ACBL-pg-req-wr-locked messages, as is appropriate.

Note that a request for a commit by a transaction does not cause aborts and is granted by the server. Also, in case of conflicts on pages or objects, the server invokes a deadlock detection algorithm and if a deadlock is detected, victim is chosen and aborted.

III. Assumptions and Requirements

Assumptions are presented first, followed by the requirements.

A. *Assumptions*

Each client cache supports only one of the caching algorithms while the server knows which cache clients are using which caching algorithms. While a client uses one of the caching algorithms it is not aware that there are clients that may use different types of caching algorithms.

Pages have version IDs (eg., timestamps) so that a server can determine whether a page in a transaction's read or write set is stale, i.e., that a page has been updated at the server by some other transaction since it has been cached by the client.

A page has a number of objects. For simplicity, it is assumed that an object does not span a page. Removal of this assumption can be made without much complexity. Each client keeps track of objects on a page so that it knows whether they have been written to or read locally (and thus are write or read locked). For simplicity and without loss of generality, it is assumed that the server deals with one client request at a time and that a client executes only one transaction at a time. It is assumed that a client has sufficient space to store the working set of a transaction. When a transaction commits, pages in its working set (read and write sets) are not purged from memory but are accessible to subsequent transactions executing on the client. Consequently, when a page is evicted from the client's cache, it is assumed that the current transaction has not accessed such a page.

We concentrate only on transactional caching as defined in [Franklin 1997], that is we are only dealing with the issues of consistent access to cached data and concurrency control and we do not address problems dealing with logs and updates of the server's pages. For example, we do not address how update of server pages is performed – one option is for the client to send the server an updated (whole) page while another option is to send only updates to objects on a page that the server has to implement on its page. Similarly, we do not deal with issues stemming from recovery of various faults such as failed message delivery, which is assumed to be handled by the communication subsystem.

These assumptions are similar to those adopted in [Franklyn 1997; Ozsu 1998, Voruganti 1999].

B. Requirements

Each client supports only one of the caching algorithms and is not aware, and hence does not support, any other caching algorithm. The server must support both types of caching algorithms while still guaranteeing correct operation – it must support operations that are one-copy serializable. The interaction between the server and clients, interaction that must be supported by the server, is summarized below. Of course, the sender and receiver are known for each message.

The server's interaction with an AOCC client consists of: client's request for a page; client purging an unused page from the cache; client's request to commit a transaction; client's letting know the server that it has evicted/purged a page; server's reply to a commit request; and server's invalidation of a page(s) at the client. In AOCC, if a transaction aborts locally, the server need not be notified as all of the transaction's work is performed locally until the commit.

The server's interaction with an ACBL client consists of: client's request for a page; client purging an unused page from the cache; client's request for a write lock on a page/object; server's reply to the client's request for a write lock; server's call-backs for a lock on a page; client's reply to a call-back request; client's informing the server that it is waiting for a read/write lock on an object locked by some other transaction; client's request to commit/abort a transaction; server's reply to client's request to commit a transaction; and server's invalidation of a page(s) in the client's cache.

IV. Interoperable Server-based Cache Consistency (ISCC)

Before we present the algorithm, we discuss the design philosophy as it relates to the server handling the AOCC and ACBL clients. A FSM for server pages is presented next. The messaging interaction between the server and the AOCC and ACBL clients completes the description of our algorithm to support server-side interoperability.

A. Integrating the AOCC and ACBL Functions

Integration of the functions of the caching algorithm on the server calls for design choices that deal with conflicts and how they are resolved. If resolution requires assigning priorities, or choosing a victim, we specify how the choices are made. We adhere to the design philosophies of the AOCC and ACBL algorithms. On the one hand, the ACBL algorithm is used for applications that do not tolerate high abort rate. On the other hand, the AOCC algorithms are suitable for applications that tolerate aborts. Because the AOCC's abort cost is low and is incurred primarily on the client, it leads to high performance. Thus, in case of conflicting access between an AOCC and an ACBL client, the ACBL client has a priority. The optimistic nature of AOCC is also retained in that the client access to cached data proceeds without delays while checking for consistency/correctness is deferred until the commit phase – AOCC's design philosophy.

Having decided on priorities in case of conflicts between the ACBL and AOCC client transactions, integration of the AOCC and ACBL functions on the server is relatively straight forward with some minor difficulties. The ease of integration is facilitated by the fact that interaction between the AOCC clients and the server is simple, with the exception of the client's request to commit. AOCC client simply requests a page, purges a page, and requests a commit of a transaction. The server simple sends or invalidates pages. Because ACBL clients have higher priorities than AOCC clients in case of conflicts, AOCC clients do not functionally affect ACBL clients. In the following we shall first discuss the server states, then the server's page-states and transitions, and finally the interaction between the server and the ACBL and AOCC clients.

It should be noted that when ACBL clients/transactions attempting to access pages in a conflicting manner which, at the server, results in a wait, a deadlock detection and resolution mechanism is invoked by the server.

B. Server's Pages – States

There are four states in the ACBL server and two states on the AOCC server. They both have one common state that is Not-cached. Besides the Not-cached state, the AOCC server has just one more state Cached; because this state cannot be combined with any of the ACBL server's states, the states for the ISCC server include the states in the ACBL server plus the Cached state from the AOCC server that is renamed to Cached-AOCC-only and it denotes the state when only AOCC clients have a page in their caches. Consequently, the ISCC server has the following states:

- *Not-cached* – the page is not cached at any clients.
- *Cached-read-locked* – the page is cached in at least one or more ACBL clients that have the page locked in a shared-read mode. The page may also be cached in zero, one or more AOCC clients.
- *Exclusive* – the page is cached and (the whole page) locked in an exclusive write mode at exactly one ACBL client. The page may also be cahed at zero, one or more AOCC clients.
- *Objects-write-locked* – the page is cached at one or more ACBL clients such that one transaction at an ACBL client

has write locks on some of the page's objects. The page may be cached in AOCC clients as well.

- *Cached-AOCC-only* – the page is cached only by AOCC clients.

Recall that each client keeps track of the local state of pages and which pages and objects have been read or written by the locally executing transaction. The server also keeps track of similar information. For each page that is cached, the server not only has the set of clients where the page is cached, but also which mode it is locked in local caches. Also, for each page that is in the *Objects-write-locked* state, the server has a list of objects that are write-locked.

As there are transactions that may be waiting for access to pages or objects that are locked in an incompatible mode, each page has a list of waiting transactions with relevant information for each blocked transaction/client that includes the transaction ID, operation (read/write), and page and object IDs for which the transaction is waiting.

For each client the server has a queue of messages that are waiting to be piggybacked to the client.

We now describe the server protocol in handling interactions with the caching clients.

C. Server page state transitions

Recall that the AOCC algorithm is simple and that the server needs to deal with only three messages from the server: AOCC-page-req, request from a client for a page; AOCC-purge, a message informing the server that a page has been purged/evicted from the client's cache; and AOCC-com-req, a request by a client to commit a transaction. Messages passed by the server to the AOCC client are: AOCC-serve-pg, message that delivers a page to the client; AOCC-invalidate, used to invalidate a page at a client; AOCC-com-cmnd, used by the server as a reply to AOCC-com-req to command the client to commit a transaction; AOCC-abort-cmnd, used by the server as a reply to AOCC-com-req to command the client to abort a transaction.

To integrate the functionalities of the ACBL and AOCC servers, we have modified the ACBL server with the functionality to support the AOCC clients. In comparison to the page state-transition FSM shown in Figure 4, there is one additional state, AOCC-cached-only, as discussed in the previous subsection, and for each state there are three additional inputs due to AOCC: AOCC-page-req, AOCC-purge, and AOCC-com-req. The resulting FSM is shown graphically in Figure 5. Note that only state transitions between different states are shown. Full FSM is defined in Table 1 (located just prior to the appendix). The state transitions due to inputs that represent reception of messages from AOCC clients are bolded in Table 1. The state transitions due to inputs from ACBL clients are not bolded and are similar to those of the ACBL server.

As will be seen shortly, the messages passed between the server and AOCC clients are relatively straight-forward in that they are very close to the original AOCC algorithm. The main difference is in the server's actions when committing. As expected, if the committing transaction has read or written stale pages then it is aborted as the server does not keep track of the read and write sets of already committed transactions. However, unlike in a pure AOCC environment, here complications arise if the committing transaction has read or written to pages that are currently locked by ACBL transactions. Since we avoid aborting ACBL clients and also avoid delaying AOCC clients, the committing AOCC transaction is aborted in case of conflict. Consider an AOCC transaction To that is being committed. If it (To) has written to a page read-locked by an ACBL transaction Tb, the AOCC transaction To is aborted because in the serialization order Tb has occurred before To, but the still executing transaction Tb could in future read pages in the write set of To. If Tb is not to be affected by the AOCC transaction To and To should not be delayed, the AOCC transaction has to be aborted. Similar reasoning applies when To has read a page that is write-locked by Tb and To is a write transaction – To is aborted because To occurred before Tb but Tb, which is still executing, could read a page written by To. Note that if To has written to pages written to by Tb but the read-set of To does not intersect with the write set of Tb and vice-versus, then To could be committed. However, since the server assumes that a transaction that has a write lock on a page/object may have also read it (it does not know otherwise), the ACBL's transaction's write-set is also a part of its transaction's read set.

In summary, for a committing AOCC transaction, if a page in its read set is write-locked by an ACBL transaction, or if a page in the write-set is read-locked or write-locked by an ACBL transaction, then the AOCC transaction is aborted.

When an aborted transaction is restarted, it is likely that it will require the same pages as when it was aborted. If a transaction was aborted because it has read a stale page, then that stale page will be invalidated and the transaction will re-fetch the page from the server while the server will deliver it without delays. Similarly, when a transaction is restarted because it has read a page that is write-locked, when the transaction holding the write-lock terminates, the lock will be released and the page will be accessible.

The situation is more subtle if a transaction is aborted because it has written to a page that is read-locked by an ACBL transaction. Recall that if a page is cached by an ACBL client then the server assumes that it is read-locked even if it is not actually accessed by any transaction at that client and the page is in the Cached-not-locked state. If an AOCC transaction To is aborted because it has written to a page P that is cached at an ACBL client Cb and, upon its restart, it writes to the same page P, then it will be aborted again if there is no activity on that page P at the client Cb where it is Cached-not-locked state. The server invalidates such pages; that is, when an AOCC transaction is aborted because it has written to a page P that on the server is the Cached-read-locked state, then the page is invalidated on the ACBL servers while on the server it still remains in the Cached-read-locked state. When an ACBL client receives the invalidation, if it has read the page, the page moves to Not-cached-read-locked state. If the transaction reads an object on P again it has to be re-fetched. If the transaction terminates without re-reading the page, the client cache moves the page locally to Not-cached and informs the server of this fact by piggybacking the information that P was purged together with its termination message (ACBL-com-req or ACBL-abort-req).

D. *Server's interaction with ACBL clients*

Interaction between the server and the ACBL clients, in the absence of AOCC clients, is of course, the same as for the pure ACBL server. In fact, because of the design philosophy that we have adopted in that the ACBL clients have a priority in resource (page and lock) acquisition over the AOCC clients, presence of AOCC clients does not affect the interaction between the ISCC server and the AOCC clients with the exception of the one subtle case just described in the previous subsection.

In the following, unless stated otherwise, any reference to a state of a page refers to the state of the page on the server.

- *ACBL-pg-req-rd-locked*: client-to-server message containing the transaction ID, Page ID, and object ID. This message is sent by the client if the transaction issues a read operation on a page that is not cached, i.e., on a page that is in Not-cached or Not-cached-read-locked states on the client. If the page, or an object on the page, is not write locked, then the page is served by sending it a ACBL-serve-pg-rd-locked message and the page moves to the Cached-read-locked state.
- *ACBL-pg-req-wr-locked*: client-to-server message containing the transaction ID, Page ID, and object ID. This message is sent by the client if the transaction issues a write operation on a page that is not cached. If the page is in Not-cached or Cached-AOCC-only states, the server sends the page to the requesting client by sending it the ACBL-serve-pg-wr-locked message indicating an exclusive write-lock on the page and the page state moves to Exclusive. If the page is in the Cached-read-locked state, the server sends a ACBL-callb message to all other ACBL clients where the page is cached. Only when all callback replies are received does the server send the requesting client the page using the ACBL-serve-pg-wr-locked message indicating that the object is write-locked; the page moves to the Objects-wr-locked state.
- *ACBL-purge*: client-to-server piggybacked message letting the server know that an unused page (not read or written by a local transaction) was purged from the client's buffer pool.
- *ACBL-rd-lock-req*: client-to-server piggybacked message requesting a read-lock for an object on a page that is locked for write by another transaction. The message includes the transaction ID, page ID, and object ID. The message is sent so that the server would perform deadlock detection and was aware of the local wait. When the transaction holding the write-lock aborts, the read lock is granted by sending the ACBL-rd-lock-grant. If the transaction holding the write-lock commits, the server sends the requested read-lock and the page to the requesting transaction using the ACBL-serve-pg-rd-locked message.
- *ACBL-rd-lock-grant*: server-to-client message that contains the page ID, object ID, and transaction ID, and it indicates to the client that a read-lock on the object is granted. This is a reply to the client's ACBL-rd-lock-req message issued when the page, or an object on the page, that the transaction wants to read is write-locked. If the transaction holding the write-lock aborts, this message is sent to the client requesting the read-lock.
- *ACBL-wr-lock-req*: client-to-server message requesting a lock escalation. The message includes the transaction ID, page ID, and object ID. If the page, or an object on a page is already write-locked by another transaction (at another client), the request is queued at the server and served when the transaction holding the lock terminates. If the requesting transaction is the only ACBL transaction having the only copy of the page, the server grants the exclusive lock on the page. It sends the client a ACBL-wr-lock-grant message indicating exclusive lock and the page moves to the Exclusive state on the server. Otherwise, the server issues ACBL-callback messages to ACBL clients where the page is cached. When all callback replies are received, a write-lock on the object is given by sending the client ACBL-wr-lock-grant indicating a write-lock on object and the page moves to the Objects-wr-locked state.
- *ACBL-wr-lock-grant*: server-to-client message that contains the page ID, object ID, and transaction ID, and it indicates to the client that the write-lock on the object is granted. This is a reply to the client's ACBL-write-lock-request.
- *ACBL-callb*: server-to-client callback message that contains the page ID, object ID, and transaction ID. It is send by the server to the client to request a write lock on the specified object. If the object was not read locally then the lock is granted by sending the server a ACBL-callb-reply message. If the page was not accessed at the client (it is in the Cached-not-locked state at the client), the page is purged and it moves to the Not-cached state (on the client); otherwise it moves to the Objects-wr-locked-foreign state (on the client). In either case, the server is informed of the outcome in the callback reply message.
- *ACBL-callb-reply*: client-to-server message that includes the page ID and object ID indicating that the call-back was successful. The message also indicates whether at the client the page was invalidated.
- *ACBL-abort-req*: client-to-server message requesting abort of a transaction to which the server responds with a ACBL-abort-cmnd message. If the transaction is holding any write locks for which other clients/transactions are waiting then when the locks of the aborting transactions are released, a waiting client that has requested the lock by using the ACBL-rd-lock-req message is granted the lock with a ACBL-rd-lock-grant message, while a client that has requested a page by using ACBL-pg-rq-rd-locked message is sent the page with the ACBL-serve-pg-rd-locked. Finally, if clients are not waiting to read the page but a writer is waiting, then the write-lock is given by

using the ACBL-serve-pg-wr-locked or ACBL-wr-lock-grant, as is appropriate.

- *ACBL-com-req*: client-to-server message, requesting the commit of a transaction identified by a transaction ID, to which the server responds with a ACBL-com-cmnd message. If the transaction is holding any write locks for which other clients/transactions are waiting then when the locks of the aborting transactions are released, a waiting client that has requested a read lock by using either of the ACBL-rd-lock-req or ACBL-pg-rq-rd-locked messages is granted the lock together with the page by using the ACBL-serve-pg-rd-locked message. If clients are not waiting to read the page but a writer is waiting, then the write-lock is given by using the ACBL-serve-pg-wr-locked message.
- *ACBL-com-cmnd*: server-to-client message indicating to the client that the commit was successful at the server and thus commanding the client to commit the transaction locally.
- *ACBL-abort-cmnd*: server-to-client message commanding the client to abort transaction.

E. *Server's interaction with AOCC clients*

The server's interaction with the AOCC clients is straight-forward with the exception of the transaction's commit.

- *AOCC-page-req*: client-to-server message identifying the requested page. The server replies with the requested page without delay even if the page is locked in an exclusive mode by an ACBL client. This is recognizing that AOCC client's abort cost is absorbed by the client and the effects on the system performance in terms of other clients and the server are minimal.
- *AOCC-serve-page*: server-to-client message used by the server to send a page to an AOCC client. If the page was originally in the Not-cached state then it moves to Cached-AOCC-only state, otherwise there is no page-state transition.
- *AOCC-purge*: client-to-server *piggybacked* message letting the server know that an unused page was purged from memory. The server simply updates its internal structures to reflect the fact that the page is no longer cached at that client. Also, if there is a message queued for piggybacking to invalidate the page at that client then that message is deleted.
- *AOCC-invalidate*: server-to-client *piggybacked* message to invalidate a page. The client invalidates the page and aborts a transaction if it has read the page.
- *AOCC-commit-request*: client-to-server message requesting the commit of a transaction. The message contains the transaction ID and the transaction's read and write sets. Each page in the read/write set has its version number. The server checks the version numbers in the read-set of the transaction with its version numbers.

  If the committing transaction has accessed stale pages then it is aborted by sending the client a AOCC-abort-cmnd message. (Invalidation of stale pages on the client should already be queued for piggybacking or be in transit). The transaction is also aborted if any of its read-set pages are write-locked or if any of its write-set pages are read or write locked. Furthermore, any pages in the transaction's write set that are read-locked at other clients are invalidated; the reason has been discussed at the end of subsection C.

  Otherwise, the transaction is committed. Each page in the transaction's write set is invalidated at each client but the client of the committing transaction.
- *AOCC-commit-cmnd*: server-to-client message sent as a reply to the transaction's request to commit; it commands the client to commit the transaction .
- *AOCC-abort-command*: server-to-client message commanding an abort of a transaction at the AOCC client. The server generates the message to abort a transaction as a reply to the client's request to commit that cannot be satisfied due to conflicts on pages with other transactions.

F. *Discussion*

The design of the interoperable server gives priorities to the ACBL clients over the AOCC ones. This was done under the assumption that makes AOCC algorithm attractive in that the cost of an abort of an AOCC client is born primarily by the client without much effect on the rest of the system. If only AOCC clients are present, a transaction is aborted only if it has accessed stale data. Even in such a situation, there is a problem that an AOCC transaction may be repeatedly aborted and hence starved. The problem here, however, is exacerbated in presence of ACBL transactions. When an AOCC transaction is aborted due to access to pages that are locked by ACBL transactions, the chances of repeated abort upon the transaction restart are higher when ACBL transactions are interactive and of long duration. However, considering the fact that ACBL algorithm is used for interactive transactions that should have a low abort rate while AOCC algorithm is used for transactions that can tolerate high abort rate this is deemed to be a reasonable approach.

V. Related Work

There is extensive work on coherency of cached data in different environments, for instance [Stenstrom 1990 (multiprocessors), Eggers 1991 (single-bus shared memory multiprocessors), and Franklin 1997 (transactional database), Zheng 2002 (mobile)]. DSM is one of the early environments for which coherency and concurrency control mechanisms were integrated [Hsu 1989, Jutla 1993]. Coherence control has also been integrated for efficiency purposes with recovery [Voruganti 1999, Bodorik 1999, Morin 2000]. Work in web DB processing that utilizes client-server architecture can be found in [Candan 2001, Challenger 1999, Datta 2001]. Middle-tier database caching in IBM and Oracle products are described in [Luo 2002, Anton 2002]. The project on Performance and Caching in Middleware Systems at IBM Watson Labs [Degenaro, 2000, 2001; Iyengar, 1999] included development of a General Purpose Cache (GPS) used to store objects, results of queries processed by a DB, in memory, or disk, or both. A GPS is targeted to support application-level caching within a middleware system based on the Accessible Business Rules (ABR) for IBM's WebSphere. Objects are results of queries executed on base tables but may also be derived/comprised of other objects. Consequently, when a base table is updated, the GPS must efficiently invalidate all objects formed directly or indirectly from such a table. The GPS also provides for purging objects from the cache based on their age.

Caching when accessing OODBs or relational OODBs has also received a lot of attention [Carey 1991, Wang 1991, Franklin 1994, Chang 1994, Carey 1994, Agrawal 1994, Adya 1995, Chang 1997, Franklin 1997, Ozsu 1998, Voruganti 1999]. The review of cache coherence algorithms in [Franklin 1997] provides taxonomy of protocols, and evaluation of several protocols under different scenarios. Good overview of interrelated problems of cache consistency, concurrency control, updating, and recovery can be found in [Voruganti 1999]. The authors propose architecture for a server that dynamically serves pages and/or objects on pages and deals with issues of data transfer, updating, and recovery.

Closest to our work is [Franklin 1994] that presents ACBL, [Adya 1995] that deals with AOCC, and [Ozsu 1998, Voruganti 1999] that deal with AACC. As was already mentioned earlier, AACC algorithm improves on AOCC and ACBL and thus addresses the problem of requiring avoidance based algorithm because high abort rate, which can be exhibited by AOCC algorithm, is not acceptable, and thus lower performance ACBL algorithm is used. Experiments simulation experiments were conducted in [Ozsu 1998 and Voruganti 1999] to examine the performance of various algorithms and approaches under different scenarios in which loads and resources were varied. It was concluded that using objects, instead of pages, as granularity of data unit for various purposes, such as data transfer and updates, is advantageous, while under different circumstance using a page as granularity of data is preferred. Also, it was observed that ACBL is actually better in situations where AOCC was thought to be superior – the difference was that more detailed simulation was conducted for a more detailed environment.

VI. Summary and Conclusions

It has been generally accepted that the AOCC algorithm has a better overall performance than the ACBL algorithm. Yet, ACBL or its variants are used in practice because AOCC may lead to high abort rate unacceptable to some transactions, such as interactive ones. We present a server-side caching algorithm that supports both ACBL and AOCC clients. This would be beneficial particularly in a situation where there are two groups of transactions that access different areas of the DB such that one gourp requires the ACBL algorithm, because a high abort rate is unacceptable, while the other group prefers the AOCC algorithm for performance. The interoperable server is also preferred in a situation where there are already existing client caches using say the ACBL algorithm, but we client(s) that use the AOCC algorithm need to be added.

We are currently conducting simulation experiments that, for various simulated environments and work-loads, compare the performance of the server to the performance of using AOCC, ACBL, and AACC caching methods. We are also examining interoperable caching servers in the middleware environment in which not only data shipping but also query shipping caches exist [Kossmann 2000].

References

[Adya 1995] Adya, A., Gruber, R., Liskov, B., and Maheshawari, U. Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks. In *ACM SIGMOD Conference Proceedings*, 1995.

[Agrawal 1987] Agrawal, R., Carey, M., and Livny, M. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM TODS,* December 1987.

[Anton 2002] Anton J., Jacobs, L., Parker J., Zeng Z., Zhang T., Web Caching for DB Applications with Pracle Web Cache, ACM SIGMOS, 2002, pp. 594-599.

[Bodorik 1998] Bodorik, P. and Jutla, D. N., "Multi-view Memory Support to Operating Systems in Locking for Transaction and Database Systems," *The Computer Journal*, Vol. 41, No. 2, 1998, 84-97.

[Bodorik 1999] Bodorik P., Jutla D., Agarwal A., "Recoverable Virtual Memory through the MultiView Computer System", Hawaii International Conference on System Sciences, Maui, Jan 5-8, 1999.

[Candan 2001], K.S.Candan , Wen-Syan Li , Qiong Luo , Wang-Pin Hsiung , Divyakant Agrawal, Enabling dynamic content caching for database-driven web sites, Proceedings of the 2001 ACM SIGMOD international conference on Management of data, p.532-543, May 21-24, 2001, Santa Barbara, California, United States.

[Carey 1991] Carey, M., Franklin, M., Livny, M., and Shekita, E. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proceedings of ACM SIGMOD Conference,* 1991.

[Carey 1994] Carey, M., Franklin, M., and Zahaariodakis, M. Fine Grained Sharing in a Page Server OODBMS. In *Proceedings of ACM SIGMOD Conference,* 1994.

[Challenger 1999] Challenger J., , A. Iyengar, P. Dantzig., "A Scalable System for Consistently Caching Dynamic Web Data", Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM'99, New York, 1999.

[Chang 1997] Chung, I., Lee, J., and Hwang, C. A Contention Based Dynamic Consistency Maintenance Scheme for Client Cache. In *Proceedings of CIKM Conference,* 1997.

[Datta 2001], A. Datta, K. Dutta, H. Thomas, D. VanderMeer, K. Ramamritham, D. Fishman. A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration. In Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[Degenaro 2000] Degenaro L., Iyengar,A., Lipkind I., "A Middleware System Which Intelligently Caches Query Results", Proceedings of ACM/IFIP Middleware 2000, Palisades, New York, April 2000.

[Degenaro 2001] Degenaro L., Iyengar, A., and Rouvellou, I. "Improving Performance with Application-Level Caching", Proceedings of the SSGRR 2001 International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, L'Aquila, Italy, August 2001.

[Eggers 1991] S. J, Eggers. Simplicity versus accuracy in a model of cache coherency overhead, IEEE Transactions on Computers, Vol. 40, No. 8, August 1991.

[Hsu 1989], Hsu M. and Tam V., Transaction Synchronization in Distributed Shared Virtual Memory Systems, Proc. Of the 13th Annual Int. Computer and Software Applications Conference, COMPSAC, September, 1989.

[Franklin 1994] Franklin, M., and Carey, M. Client-Server Caching Revisited. In *Distributed Object Management.* Edited by T. Ozsu, U. Dayal, and P. Valduriez. Morgan Kaufmann, 1994.

[Franklin 1997] Franklin M.J., "Transactional Client-Server Cache Consistency: Alternatives and Performance", ACM Transactions on Database Systems, Vol. 22, No. 3, 1997, pp. 315-363.

[Iyengar 1999] Iyengar A., "Design and Performance of a General-Purpose Software Cache", Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference (IPCCC'99), Phoenix/Scottsdale, Arizona, February 1999.

[Jutla 1993] Jutla D.N., Bodorik P., Riordon, J.S., "Integrated Concurrency-Coherence Control in Distributed Shared Memory, Fifth International Conference on Computing and

[Kossman 2000] Kossman D., Franklin M.J., Drasch, G., Cache Investment: Integrating Query Optimization and Distributed Data Placement, ACM Transactions on Database Systems, Vol. 25, No. 4, December 2000, pages 517-558.

[Luo 2002] Luo Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F. Middle-Tier Database Caching for e-Business, ACM SIGMOD, Madison, Wisconsin, 2002, pp. 600-611.

[Morin 2000] Morin C.,Kermarrec, A-M.,, Banatre M., Gefflaut, A., "An Efficient and Scalable Approach for Implementing Fault-Tolerant DSM Architectures," IEEE Transactions on Computers, Vol. 49, No. 5, May 2000., pp. 414-429.

[Ozsu 1998] Ozsu M.T., Voruganti, K., Unrau, R.C. "An Asynchronous Avoidance-Based Cache Consistency Algorithm for Client Caching DBMSs", Proc. 24th VLDB Conf., New York, 1998, pp. 440-451.

[Stenstrom 1990] Stenstrom P., A Survey of Cache Coherence Schemes for Multiprocessors, IEEE Computer. June 1990.

[Voruganti 1999] Voruganti, K., Ozsu, T., and Unrau, R. An Adaptive Hybrid Server Architecture for Client Caching Object DBMSs. In *Proceedings of 25th VLDB Conference,* 1999, pp. 150-161.

[Wang 1991] Wang, Y., and Rowe, L. Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In *Proceedings of ACM SIGMOD Conference,* 1991.

[Zheng 2002] Zheng B., Xu, J., Lee, D.L., Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments, IEEE Transactions on Computers, Vol. 51, No. 10, October 2002, 1141-1153.

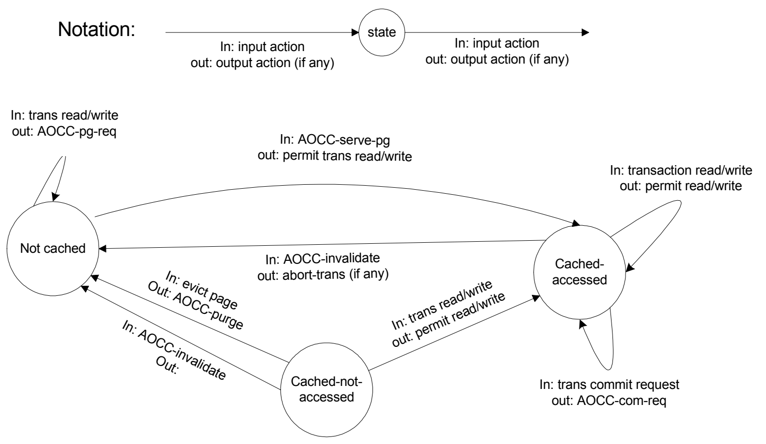


Figure 1. AOCC Client — State Transition Diagram for a Page

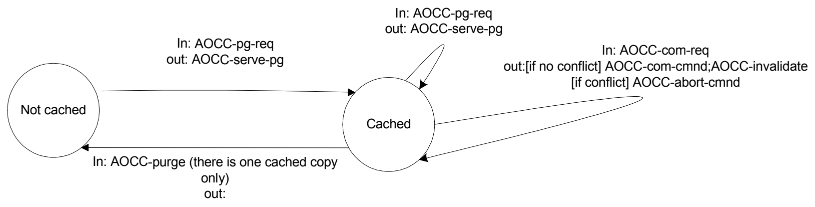
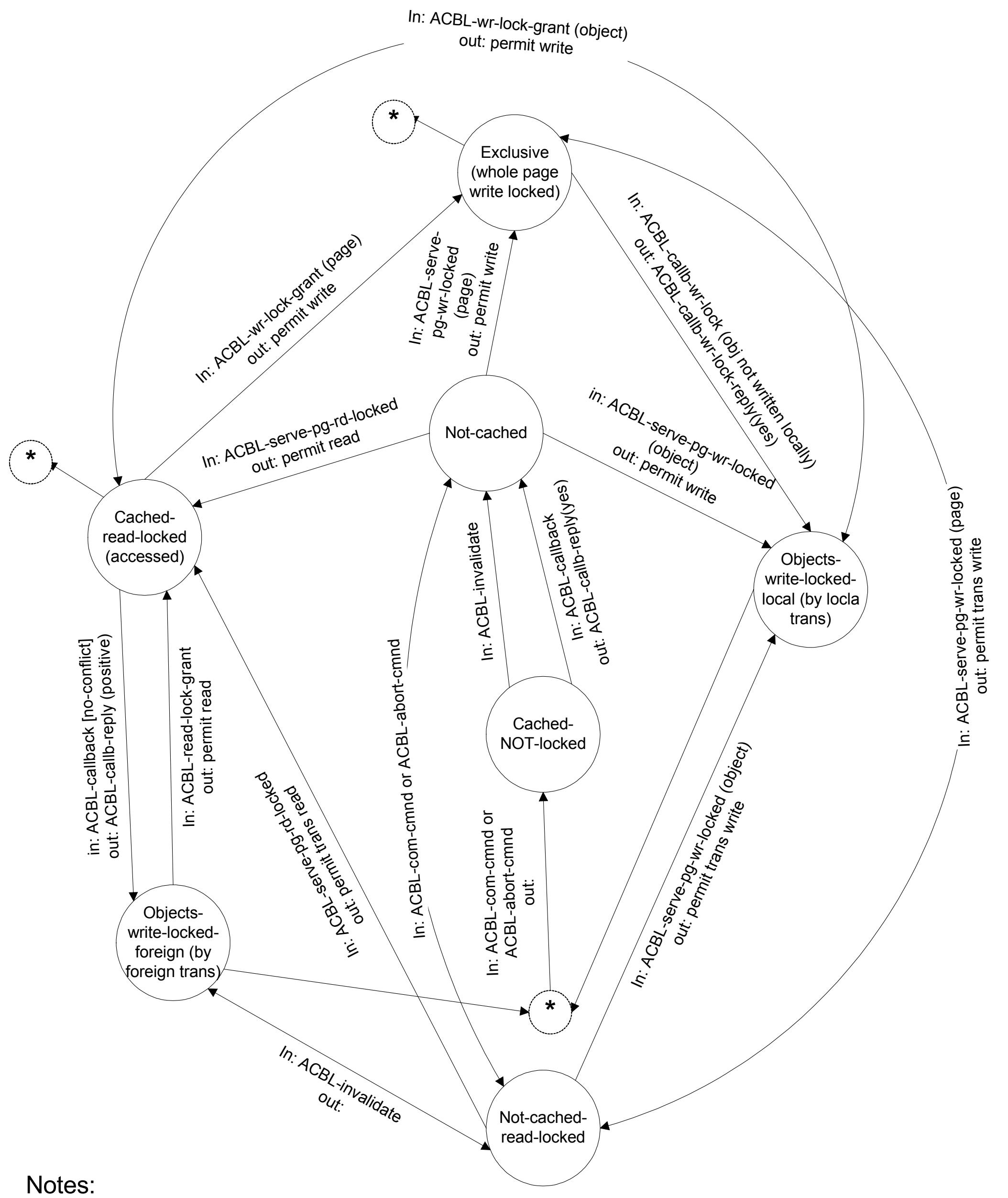

Not cached
Cached
In: AOCC-pg-req
out: AOCC-serve-pg
In: AOCC-pg-req
out: AOCC-serve-pg
In: AOCC-com-req
out:[if no conflict] AOCC-com-cmnd;AOCC-invalidate
[if conflict] AOCC-abort-cmnd
In: AOCC-purge (there is one cached copy only)
out:

Figure 2. AOCC Server — State Transition Diagram for a Page


In: ACBL-wr-lock-grant (object)
out: permit write
*
Exclusive (whole page write locked)
In: ACBL-wr-lock-grant (page)
out: permit write
In: ACBL-serve-pg-wr-locked (page)
out: permit write
In: ACBL-callb-wr-lock (obj not written locally)
out: ACBL-callb-wr-lock-reply(yes)
Not-cached
In: ACBL-serve-pg-rd-locked
out: permit read
in: ACBL-serve-pg-wr-locked (object)
out: permit write
*
Cached-read-locked (accessed)
In: ACBL-invalidate
In: ACBL-callback
out: ACBL-callb-reply(yes)
Objects-write-locked-local (by locla trans)
In: ACBL-serve-pg-wr-locked (page)
out: permit trans write
in: ACBL-callback [no-conflict]
out: ACBL-callb-reply (positive)
In: ACBL-read-lock-grant
out: permit read
In: ACBL-serve-pg-rd-locked
out: permit trans read
In: ACBL-com-cmnd or ACBL-abort-cmnd
Cached-NOT-locked
In: ACBL-com-cmnd or ACBL-abort-cmnd
out:
In: ACBL-serve-pg-wr-locked (object)
out: permit trans write
Objects-write-locked-foreign (by foreign trans)
*
In: ACBL-invalidate
out:
Not-cached-read-locked

Notes:

* Trans read and trans write represent read and write operations issued by a local transaction.
* Inputs that do not lead to state changes are NOT shown.

* … State transition to the Cached-NOT-locked state upon reception of ACBL-com-cmnd or ACBL-abort-cmnd from the server.

Figure 3. ACBL Client — State Transition Diagram for a Page

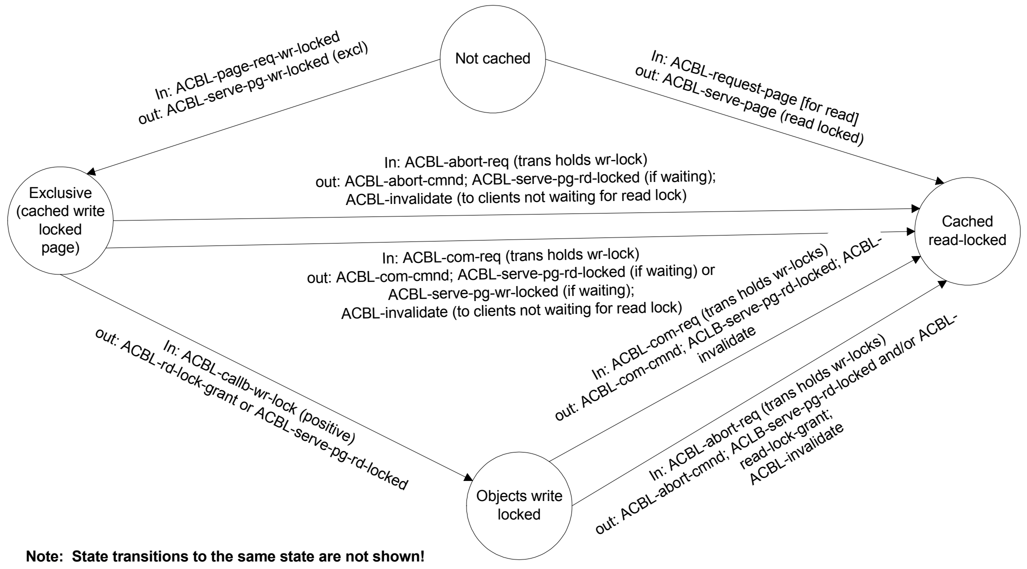

Not cached
In: ACBL-page-req-wr-locked
out: ACBL-serve-pg-wr-locked (excl)
In: ACBL-request-page [for read]
out: ACBL-serve-page (read locked)
Exclusive (cached write locked page)
Cached read-locked
In: ACBL-abort-req (trans holds wr-lock)
out: ACBL-abort-cmnd; ACBL-serve-pg-rd-locked (if waiting);
ACBL-invalidate (to clients not waiting for read lock)
In: ACBL-com-req (trans holds wr-lock)
out: ACBL-com-cmnd; ACBL-serve-pg-rd-locked (if waiting) or
ACBL-serve-pg-wr-locked (if waiting);
ACBL-invalidate (to clients not waiting for read lock)
In: ACBL-callb-wr-lock (positive)
out: ACBL-rd-lock-grant or ACBL-serve-pg-rd-locked
In: ACBL-com-req (trans holds wr-locks)
out: ACBL-com-cmnd; ACLB-serve-pg-rd-locked; ACBL-invalidate
In: ACBL-abort-req (trans holds wr-locks)
out: ACBL-abort-cmnd; ACLB-serve-pg-rd-locked and/or ACBL-read-lock-grant;
ACBL-invalidate
Objects write locked
Note: State transitions to the same state are not shown!

Figure 4. ACBL Server — State Transition Diagram for a Page

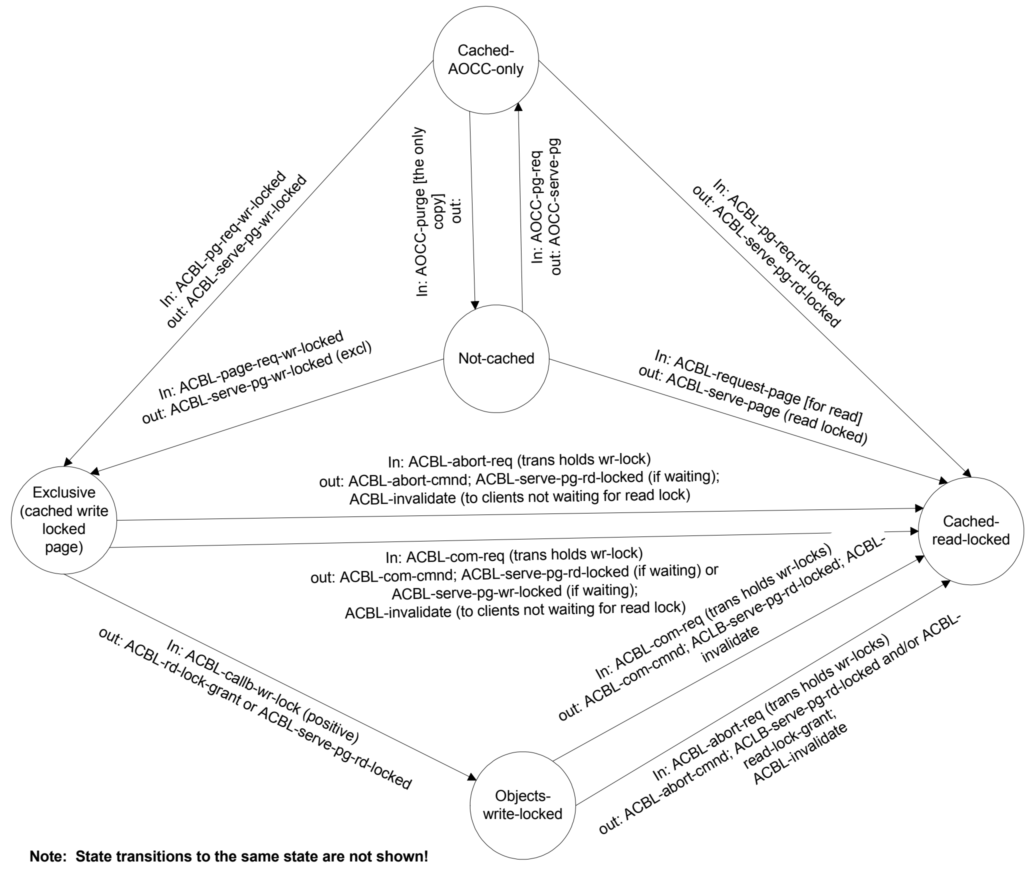

Cached-AOCC-only
In: AOCC-purge [the only copy]
out:
In: AOCC-pg-req
out: AOCC-serve-pg
In: ACBL-pg-req-wr-locked
out: ACBL-serve-pg-wr-locked
In: ACBL-pg-req-rd-locked
out: ACBL-serve-pg-rd-locked
Not-cached
In: ACBL-page-req-wr-locked
out: ACBL-serve-pg-wr-locked (excl)
In: ACBL-request-page [for read]
out: ACBL-serve-page (read locked)
Exclusive (cached write locked page)
Cached-read-locked
In: ACBL-abort-req (trans holds wr-lock)
out: ACBL-abort-cmnd; ACBL-serve-pg-rd-locked (if waiting);
ACBL-invalidate (to clients not waiting for read lock)
In: ACBL-com-req (trans holds wr-lock)
out: ACBL-com-cmnd; ACBL-serve-pg-rd-locked (if waiting) or
ACBL-serve-pg-wr-locked (if waiting);
ACBL-invalidate (to clients not waiting for read lock)
In: ACBL-callb-wr-lock (positive)
out: ACBL-rd-lock-grant or ACBL-serve-pg-rd-locked
In: ACBL-com-req (trans holds wr-locks)
out: ACBL-com-cmnd; ACLB-serve-pg-rd-locked; ACBL-invalidate
In: ACBL-abort-req (trans holds wr-locks)
out: ACBL-abort-cmnd; ACLB-serve-pg-rd-locked and/or ACBL-read-lock-grant;
ACBL-invalidate
Objects-write-locked
Note: State transitions to the same state are not shown!

Figure 5. ISCC Server — State Transition Diagram for a Page

Table 1 ISCC Server -- State transition diagram for a page

state	input	condition	new state	output	description
Not-cached	ACBL-pg-req-rd-locked		Cached-rd-locked	ACBL-serve-pg-rd-locked	req for a pg with a rd-lock
	ACBL-pg-req-wr-locked		Exclusive	ACBL-serve-pg-wr-locked(excl)	req for a pg with a wr-lock (exclusive or object)
	AOCC-pg-req		**Cached-AOCC**	**AOCC-serve-pg**	**Page request served**
Cached-rd-locked	ACBL-pg-req-rd-locked			ACBL-serve-pg-rd-locked	Request for a page
	ACBL-pg-req-wr-locked		--	ACBL-callbs	Send callbs for wr-lock
	ACBL-purge	1 cached only	Not-cached		The only cached copy was purged
		>1 cached	--		Note in data structures
	ACBL-wr-lock-req		--	ACBL-callbs	Send callbs to cached copies for wr-lock
	ACBL-callb-reply	all replies received	Objects-wr-locked	ACBL-wr-lock-grant or ACBL-serve-page-wr-locked (exl or object)	All callbs replies received - either (grant wr-lock on object or the whole page) or (serve page with a write lock on page or object)
	ACBL-com-req	waiting callback	Objecs-wr-locked	ACBL-com-cmnd; ACBL-wr-lock-grant	Commit cmnd to requesting client; send reply to waiting client
		no waiting callb	--	ACBL-com-cmnd	Send commit command
	ACBL-com-req	waiting callback	--	ACBL-com-cmnd; ACBL-wr-lock-grant	Abort cmnd to requesting client;sent callback reply to waiting client
		no waiting callb	--	ACBL-abort-cmnd	Abort cmnd to requesting client
	AOCC-page-req		**--**	**AOCC-serve-pg**	**Page request by AOCC client served**
	AOCC-purge		**--**		**Note in data structures**
	AOCC-com-req	**no conflict**	**--**	**AOCC-com-cmnd; AOCC-invalidate; ACBL-invalidate**	**Commit trans. Transaction's write-set must be invalidate at other clients.**
		conflict	**--**	**AOCC-abort-cmnd; ACBL invalidate**	**AOCC trans aborted; page invalidated**
Exclusive	ACBL-pg-req-rd-locked		--	ACBL-callb-wr-lock	Send callbac for wr-lock
	ACBL-pg-req-wr-locked		--		Req by trans not holding wr-lock; blocked
	ACBL-purge		--		Note in data structures
	ACBL-rd-lock-req				N/A- only one copy exists
	ACBL-wr-lock-req		--		Request blocked
	ACBL-callb-wr-lock-reply		Objects-wr-locked	ACBL read-lock-grant or ACBL-serve-page-rd-locked	Client released exclusive wr-lock on page; page has objects write-locked with the rest of objects read-locked
	ACBL-com-req	has write locks	Cached-read-locked	ACBL-com-cmnd; ACBL-serve-pg-rd-locks or ACBL-serve-pg-wr-locked; ACBL-invalidate	Commit cmnd to requesting client; send page read-locked to waiting clients; invalidate page if read-lock not requested
	ACBL-abort-req	has write locks	Cached-read-locked	ACBL-abort-cmnd; (ACBL-serve-pg-rd-locked and/or ACBL-rd-lock-grant) or (ACBL-wr-lock-grant or ACBL-serve-pg-wr-locked)	Abort cmnd to requesting client; send read-locks to waiting clients
	AOCC-page-req		**--**	**AOCC-serve-pg**	**Page request by AOCC client served**
	AOCC-purge		**--**		**Note in data structures**
	AOCC-com-req	**no conflict**	**--**	**AOCC-com-cmnd; AOCC-invalidate; ACBL-invalidate**	**Commit trans. Transaction's write-set must be invalidate at other clients.**
		conflict	**--**	**AOCC-abort-cmnd**	**AOCC trans aborted; page invalidated**
Objects-wr locked	ACBL-pg-req-rd-locked	conflict	--		Object already wr-locked - blocked
		no conflict	--	ACBL-serve-pg-rd-locked	Read on a reqeusted page does not conflict with object write-locks
	ACBL-pg-req-wr-locked		--		Request blocked
	ACBL-purge		--		Note in data structures
	ACBL-rd-lock-req		--		Request blocked
	ACBL-wr-lock-req	not have wr locks	--		Page write-locked; request blocked
		has write locks	--	ACBL-callbs	Send callbs to cached copies-wait
	ACBL-callb-reply	all positive	--	ACBL-wr-lock-grant	All callbs positive - grant wr-lock
		negative replies	--		Negative reply(ies) - req blocked
	ACBL-com-req	trans has write-locks	Cached-read-locked	ACBL-com-cmnd; ACBL-serve-pg-rd-lock; ACBL-invalidate	Commit cmnd to requesting client; send page read-locked to waiting clients; invalidate page if read-lock not requested
	ACBL-abort-req	trans has write-locks	Cached-read-locked	ACBL-abort-cmnd; ACBL-rd-lock-grant; ACBL-serve-pg-rd-lock	Abort cmnd to requesting client; read-lock sent to waiting clients; page read-locked sent to waiting clients
	AOCC-page-req		**--**	**AOCC-serve-pg**	**Page request by AOCC client served**
	AOCC-purge		**--**		**Note in data structures**
	AOCC-com-req	**no conflict**	**--**	**AOCC-com-cmnd; AOCC-invalidate; ACBL-invalidate**	**Commit trans. Transaction's write-set must be invalidate at other clients.**
		conflict	**--**	**AOCC-abort-cmnd; ACBL invalidate**	**AOCC trans aborted; page invalidated**
Cached-AOCC-only (only at AOCC clients)	ACBL-pg-req-rd-locked		Cached-rd-locked	ACBL-serve-pg-rd-locked	req for a pg with a rd-lock
	ACBL-pg-req-wr-locked		Exclusive	ACBL-serve-pg-wr-locked(excl)	req for a pg with a wr-lock (exclusive or object)
	AOCC-page-req		**--**	**AOCC-serve-pg**	**Page request by AOCC client served**
	AOCC-purge	**1 cached only**	**Not-cached**		**The only cached copy was purged**
		>1 cached	**--**		**Note in data structures**
	AOCC-com-req	**no conflict**	**--**	**AOCC-com-cmnd; AOCC-invalidate**	**Commit trans. Transaction's write-set must be invalidate at other clients.**
		conflict	**--**	**AOCC-abort-cmnd**	**AOCC trans aborted**

Notes: N/A … not applicable to the page in this state

ACBL-com-req, and ACBL-abort-req are not shown as inputs if they have no effect on the page.

(There is no page transition but corresponding ACBL-com-req and ACBL-abort-req messages/requests are sent to the server.)

Similarly to ACBL-com-req and ACBL-abort-req, inputs that do not have effect on the page are not shown.

APPENDIX

Table A-1 AOCC Client -- State transition diagram for a page

state	input	condition	new state	output	description
Not-cached	AOCC-serve-pg		Cached-accessed	permit trans read/write	Server sent a page for read or write
	trans read		--	AOCC-pg-req	
	trans write		--	AOCC-pg-req	
Cached-accessed	trans read		--	permit trans read	
	trans write		--	permit trans write	
	trans commit		--	AOCC-com-req	Trans requested commit
	trans abort		--		Local trans abort
	AOCC-invalidate		Not-cached	trans abort	Abort local transaction
	AOCC-com-cmnd		--	commit transaction	Command from server to commit trans
	AOCC-abort-cmnd		--	trans abort	Command from server to abort trans
Cached-not-accessed	trans read		Cached-accessed	permit trans read	
	trans write		Cashed-accessed	permit trans write	
	evict		Not-cached	AOCC-purge	
	AOCC-invalidate		Not-cached		

Table A-2 AOCC Server -- State transition diagram for a page

state	input	condition	new state	output	description
Not-cached	AOCC-pg-req		Cached	AOCC-serve-pg	Client request for a page - served
Cached	AOCC-pg-req			AOCC-serve-pg	Client request for a page - served
	AOCC-purge	1 cached only	Not-cached		The only cached copy was purged
		>1 cached	--		Note in data structures
	AOCC-com-req	no-conflict		AOCC-com-cmnd; AOCC-invalidate	Commit trans. Transaction's write-set must be invalidate at other clients.
		conflict	--	AOCC-abort-cmnd	Trans read/write set was updated

Notes: Inputs that do not have effect on the page are not shown.

Table A-3 State transition diagram for pages on an ACBL client

state	input/operation	condition	new state	output	description
Not-cached	trans-read		--	ACBL-page-request-rd-locked	Page requested for read
	trans-write		--	ACBL-page-request-wr-locked	Page requested for write
	ACBL-serve-pg-rd-locked		Cached-rd-locked	permit trans read	Server sent page for read with a rd-lock
	ACBL-serve-page-wr-locked	page wr-lock	Exclusive	permit trans write	Server sent page with whole page wr-locked
		obj wr-lock	objs-wr-locked-local	permit trans write	Server sent page with obj wr-locked
Cached-rd-locked	trans-read		--	permit trans read	Read by a trans
	trans-write		--	ACBL-wr-lock-request	Send a request for a write lock - trans is blocked
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-wr-lock-grant	page wr-lock	Exclusive	permit trans write	Server sent wr-lock on the whole page
		object wr-lock	objs-wr-locked-local	permit trans write	Server sent wr-lock on object
	ACBL-callback	no-conflict	objs-wr-locked-foreign	ACBL-callb-reply	Wr-lock asked by foreign trans --given locally
	ACBL-com-cmnd	waiting callback	objs-wr-locked-foreign		Callback waiting for page/object
		no waiting callb	objs-wr-locked-foreign		No callback waiting
	ACBL-abort-cmnd	waiting callback	Cached-NOT-locked		Callback waiting for page/object
		no waiting callb	Cached-NOT-locked		No callback waiting
Exclusive	trans-read		--	permit trans read	Read by a trans
	trans-write		--	permit trans write	Write by a transaciton
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-callb-wr-lock	obj NOT written	objs-wr-locked-local	ACBL-callb-wr-lock-reply	Object NOT written locally - positive reply
	ACBL-com-cmnd		Cached-NOT-locked		Server-to-client cmnd to commit trans
	ACBL-abort-cmnd		Cached-NOT-locked		Server-to-client cmnd to abort trans
objs-wr-locked-local	trans-read		--	permit trans read	Read by trans
	trans-write	obj wr-locked	--	Permit trans write	Write to obj already wr-locked by the trans
		obj not wr-locked	--	ACBL-wr-lock-req	Write to obj not wr-locked; wait
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-wr-lock-grant		--	permit trans write	Server sent wr-lock on obj
	ACBL-com-cmnd		Cached-NOT-locked		Server-to-client cmnd to commit trans
	ACBL-abort-cmnd		Cached-NOT-locked		Server-to-client cmnd to abort trans
objs-wr-locked-by-foreign-trans	trans-read	obj not wr-locked	--	permit trans read	Read-- obj NOT wr-locked by foreign trans
		obj wr-locked	--	ACBL-rd-lock-request	Read-- obj IS wr-locked by foreign trans; wait
	trans-write		--	write blocked	Write by a trans -- trans is blocked
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-callback	no-conflict	--	ACBL-callb-reply-positive	Wr-lock asked by foreign trans --given locally
	ACBL-read-lock-grant		Cached-read-locked	permit trans read	Foreign trans aborted -- read-lock granted
	ACBL-invalidate		Not-cached-rd-locked		Foreign trans committed- trans read not waiting
	ACBL-com-cmnd	waiting callback	objs-wr-locked-foreign		Callback waiting for page/object
		no waiting callb	objs-wr-locked-foreign		No callback waiting
	ACBL-abort-cmnd	waiting callback	Cached-NOT-locked		Callback waiting for page/object
		no waiting callb	Cached-NOT-locked		No callback waiting
Cached-not-locked	trans-read		Cached-rd-locked	permit trans read	Read by trans on a page not rd-locked
	trans-write		--	ACBL-wr-lock-request	Write by trans - send req for wr-lock; wait
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-wr-lock-grant	page wr-lock	Exclusive	permit trans write	Server-to-client wr-lock grant on whole page
		obj wr-lock	objs-wr-locked-local	permit trans write	Server-to-client wr-lock grant on obj
	ACBL-callback		Not-cached	ACBL-callb-reply	Foreign trans asked for wr-lock on object
	ACBL-invalidate		Not-cached		
Not-cached-read-locked	trans-read		--	ACBL-page-req-read-locked	Read by trans- page is not cached but rd-locked
	trans-write		--	ACBL-page-req-wr-locked	Page requested for write
	trans-abort		--	ACBL-abort-request	Client-to-server request to abort trans
	trans-commit		--	ACBL-com-request	Client-to-server request to commit trans
	ACBL-serve-page-rd-locked		Cached-read-locked	permit trans read	Server sent page for read with a rd-lock
	ACBL-serve-page-wr-locked	page wr-lock	Exclusive	permit trans write	Server sent page with whole page wr-locked
		obj wr-lock	objs-wr-locked-local	permit trans write	Server sent page with obj wr-locked
	ACBL-callback	no-conflict	objs-wr-locked-foreign	ACBL-callb-reply	Wr-lock asked by foreign trans --given locally
	ACBL-com-cmnd	waiting callback	objs-wr-locked-foreign		Callback waiting for page/object
		no waiting callb	objs-wr-locked-foreign		No callback waiting
	ACBL-abort-cmnd	waiting callback	Cached-NOT-locked		Callback waiting for page/object
		no waiting callb	Cached-NOT-locked		No callback waiting

Notes: **N/A … not applicable to the page in this state**

ACBL-com-req and ACBL-abort-req are not shown as inputs if they have no effect on the page.

(There is no page transition but corresponding ACBL-com-req and ACBL-abort-req messages/requests are sent to the server.)

Inputs that do not have effect on the page are not shown.

Table A-4 ACBL Server -- State transition diagram for a page

state	input	condition	new state	output	description
Not-cached	ACBL-pg-req-rd-locked		Cached-rd-locked	ACBL-serve-pg-rd-locked	req for a pg with a rd-lock
	ACBL-pg-req-wr-locked		Exclusive	ACBL-serve-pg-wr-locked(excl)	req for a pg with a wr-lock (exclusive or object)
Cached-rd-locked	ACBL-pg-req-rd-locked			ACBL-serve-pg-rd-locked	Request for a page
	ACBL-pg-req-wr-locked		--	ACBL-callbs	Send callbs for wr-lock
	ACBL-purge	1 cached only	Not-cached		The only cached copy was purged
		>1 cached	--		Note in data structures
	ACBL-wr-lock-req		--	ACBL-callbs	Send callbs to cached copies for wr-lock
	ACBL-callb-reply	all replies received	Objects-wr-locked	ACBL-wr-lock-grant or ACBL-serve-page-wr-locked (exl or object)	All callbs replies received - either (grant wr-lock on object or the whole page) or (serve page with a write lock on page or object)
	ACBL-com-req	waiting callback	Objecs-wr-locked	ACBL-com-cmnd; ACBL-wr-lock-grant	Commit cmnd to requesting client; send reply to waiting client
		no waiting callb	--	ACBL-com-cmnd	Send commit command
	ACBL-com-req	waiting callback	--	ACBL-com-cmnd; ACBL-wr-lock-grant	Abort cmnd to requesting client;sent callback reply to waiting client
		no waiting callb	--	ACBL-abort-cmnd	Abort cmnd to requesting client
Exclusive	ACBL-pg-req-rd-locked		--	ACBL-callb-wr-lock	Send callbac for wr-lock
	ACBL-pg-req-wr-locked		--		Req by trans not holding wr-lock; blocked
	ACBL-purge		--		Note in data structures
	ACBL-rd-lock-req				N/A- only one copy exists
	ACBL-wr-lock-req		--		Request blocked
	ACBL-callb-wr-lock-reply		Objects-wr-locked	ACBL read-lock-grant or ACBL-serve-page-rd-locked	Client released exclusive wr-lock on page; page has objects write-locked with the rest of objects read-locked
	ACBL-com-req	has write locks	Cached-read-locked	ACBL-com-cmnd; ACBL-serve-pg-rd-locks or ACBL-serve-pg-wr-locked; ACBL-invalidate	Commit cmnd to requesting client; send page read-locked to waiting clients; invalidate page if read-lock not requested
	ACBL-abort-req	has write locks	Cached-read-locked	ACBL-abort-cmnd; (ACBL-serve-pg-rd-locked and/or ACBL-rd-lock-grant) or (ACBL-wr-lock-grant or ACBL-serve-pg-wr-locked)	Abort cmnd to requesting client; send read-locks to waiting clients
Objects-wr-locked	ACBL-pg-req-rd-locked	conflict	--		Object already wr-locked - blocked
		no conflict	--	ACBL-serve-pg-rd-locked	Read on a reqeusted page does not conflict with object write-locks
	ACBL-pg-req-wr-locked		--		Request blocked
	ACBL-purge		--		Note in data structures
	ACBL-rd-lock-req		--		Request blocked
	ACBL-wr-lock-req	not have wr locks	--		Page write-locked; request blocked
		has write locks	--	ACBL-callbs	Send callbs to cached copies-wait
	ACBL-callb-reply	all positive	--	ACBL-wr-lock-grant	All callbs positive - grant wr-lock
		negative replies	--		Negative reply(ies) - req blocked
	ACBL-com-req	trans has write-locks	Cached-read-locked	ACBL-com-cmnd; ACBL-serve-pg-rd-lock; ACBL-invalidate	Commit cmnd to requesting client; send page read-locked to waiting clients; invalidate page if read-lock not requested
	ACBL-abort-req	trans has write-locks	Cached-read-locked	ACBL-abort-cmnd; ACBL-rd-lock-grant; ACBL-serve-pg-rd-lock	Abort cmnd to requesting client; read-lock sent to waiting clients; page read-locked sent to waiting clients

Notes: Inputs that do not have effect on the page are not shown. E.g.,
ACBL-com-req and ACBL-abort-req are not shown as inputs if they have no effect on the page.
N/A … not applicable to the page in this state