DB2's use of the coupling facility for data sharing

by J. W. Josten
C. Mohan
I. Narang
J. Z. Teng

We examine the problems encountered in extending DATABASE 2™ (DB2®) for Multiple Virtual Storage/Enterprise Systems Architecture (MVS/ESA™), also called DB2 for OS/390™, an industrial-strength relational database management system originally designed for a single-system environment, to support the multisystem shared-data architecture. The multisystem data sharing function was delivered in DB2 Version 4. DB2 data sharing requires a System/390® Parallel Sysplex™ environment because DB2's use of the coupling facility technology plays a central role in delivering highly efficient and scalable data sharing functions. We call this the shared-data architecture because the coupling facility is a unique feature that it employs.

One approach to improving the capacity and availability characteristics of a single-system database management system (DBMS) is to use multiple systems. Before the introduction of the System/390* (S/390*) Parallel Sysplex*, there were two major architectures in use in the multisystem environment: the shared-disk (SDi) architecture, also called data sharing,[1] and the shared-nothing (SN) or partitioned architecture.[2] The S/390 Parallel Sysplex introduces a third multisystem architecture called the shared-data (SDa) architecture.

With SDi, all the disks containing the databases are accessible from all the sharing systems and each system has its own buffer pool (BP) to cache data in processor storage for fast reference. Every system that has an instance of the DBMS executing on it may access and modify any portion of the database on the shared disks. Because each instance has its own buffer pool and because conflicting accesses to the same data may be made from different systems, the interactions among the systems must be controlled, using various synchronization protocols. This necessitates global locking and protocols for the maintenance of buffer coherency. SDi is the approach used in IBM's Information Management System/Virtual Storage (IMS*/VS) data sharing product,[3–5] and Amoeba project,[6] and DEC's VAX** DBMS and VAX Rdb/VMS**.[7–9]

With SN, each system owns a portion of the database and only that portion may be directly read or modified by that system. That is, the database is partitioned among the multiple systems. The kind of synchronization protocols needed for SDi are not needed for SN. But a transaction that accesses data in multiple systems would need a form of two-phase commit protocol[10,11] to coordinate its activities. This is the approach taken in Tandem's NonStop** SQL[12] (System Query Language), Teradata's DBC/1012,[13] and the University of Wisconsin's Gamma.[14]

0018-8670/97/$5.00 © 1997 IBM

SDi has many advantages over SN.[1,6] Some of them are: workload balancing, horizontal growth for capacity, single-system image, and availability. Data do not have to be split across different systems for reasons of capacity or availability. SDi gives the possibility of improved availability for data and service. However, the performance penalty of intersystem message passing and increased disk access to implement the global locking and buffer coherency functions is a heavy price to pay to gain the advantages of SDi. By using the coupling facility for high-speed global locking and buffer coherency functions, the SDa architecture can deliver all the benefits of SDi without the heavy performance penalties that are otherwise suffered.

This paper describes the problems associated with and the design approaches needed for migrating an industrial-strength relational DBMS from a single-system environment to the SDa environment. We describe the design approaches taken by the DATABASE 2* (DB2*) for Multiple Virtual Storage/Enterprise Systems Architecture (MVS/ESA*) Version 4 (hereafter referred to as DB2 V4) data sharing function[15] for efficient intersystem concurrency control and maintenance of coherency among the different systems' local buffers in the SDa architecture. Use of the coupling facility (CF) is a key element of DB2's data sharing design.

The rest of the paper is organized as follows: First we give a brief overview of the DB2 SDa architecture. Next we describe the global locking problem, how DB2 data sharing uses the CF to implement fast global locking, and how "retained locks" are used to maintain data integrity across a failure of a DB2 DBMS instance. We describe the intersystem buffer coherency problem and how DB2 uses the CF to solve this problem. We also describe how DB2 logging and data recovery work with data sharing. Then we describe the DB2 design for recovering from various CF-related failures and, finally, we summarize.

A brief description of DB2 data sharing

The N-way multisystem data sharing function that was introduced in DB2 V4 provides DB2 applications with full read and write concurrent access to databases, on shared direct access storage devices (DASDs), between multiple DB2 subsystems. The DB2 subsystems may reside on the same or on different MVS images. The set of DB2 subsystems sharing the data belong to a *DB2 data sharing group*. Each DB2 subsystem is a *member* of the group. Data sharing is an optional feature of DB2 V4.

DB2 data sharing requires the services of the S/390 Parallel Sysplex. This means that an MVS sysplex must be established with MVS/ESA Version 5 or higher and at least one CF must be configured into the sysplex (two or more CFs are recommended for performance and availability reasons) and also at least one sysplex timer must be configured. (Two sysplex timers are recommended to remove the "single point of failure."[16])

As shown in Figure 1, all the members of a DB2 data sharing group must reside within a single MVS sysplex, which can contain multiple DB2 groups. Also, a nonsharing DB2 subsystem may reside within the same MVS sysplex as another DB2 data sharing group.

A transaction accesses data belonging to a DB2 group from within a single member of the group. Applications and transactions are unaware that data sharing is taking place across the DB2 group, and do not know if their particular data are being actively shared or not. DB2 automatically manages all of the multisystem concurrency and buffer coherency issues, which are transparent to the applications.

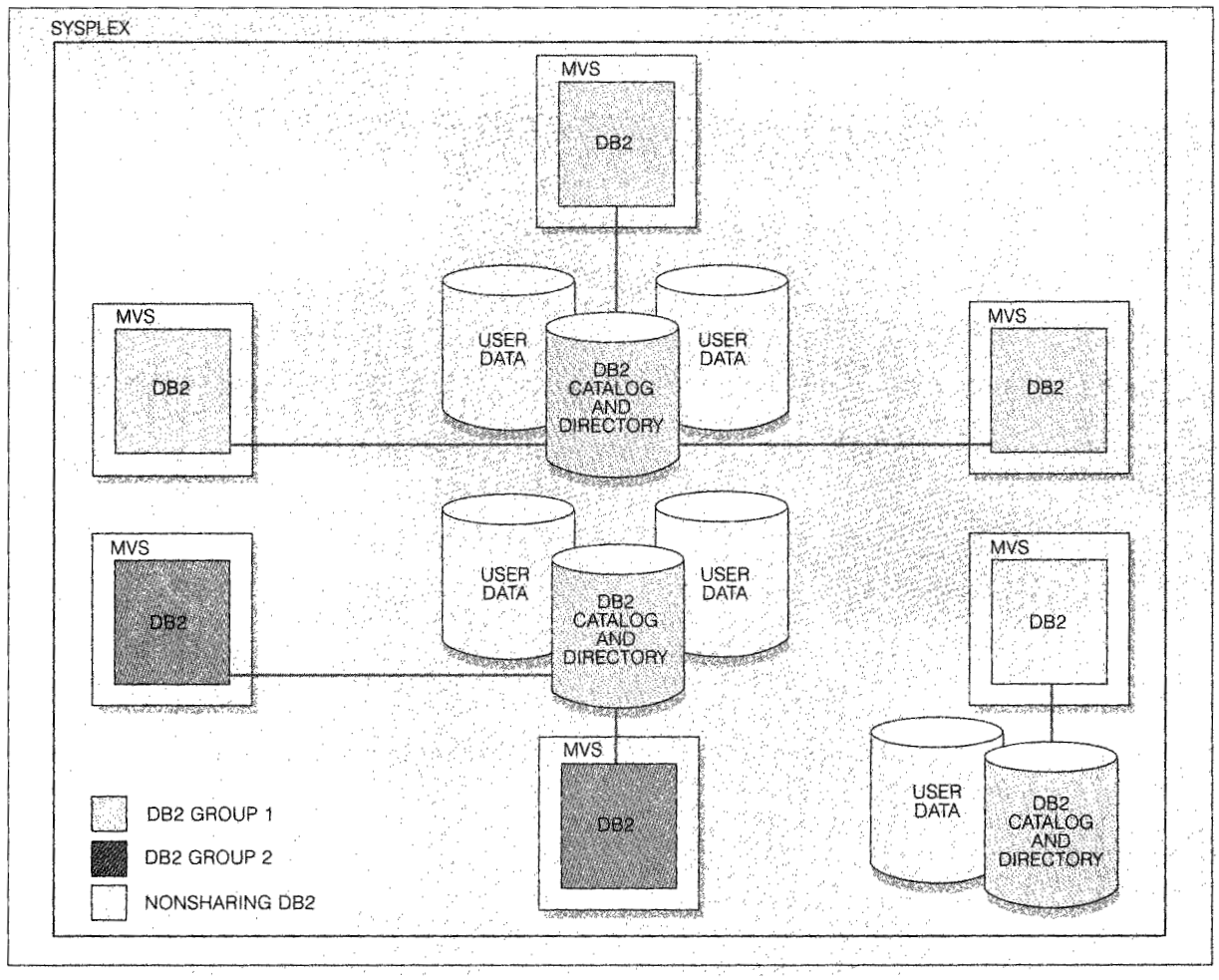
DB2 assumes that *all* data are capable of being shared across the group. Actual sharing is controlled by workload scheduling, DASD connectivity, and authorization. DB2 activates its multisystem concurrency and coherency controls only when data are *actually* shared between systems. DB2 data sharing supports data access concurrency at every level normally supported by DB2 (table space, table, page, or row).

Each member of a DB2 data sharing group must have access to shared DASD containing:

- The MVS user catalog, pointed to by the MVS master integrated catalog facility (ICF) catalog on each MVS
- A single shared DB2 catalog and directory
- Shared DB2 databases
- The ICF user catalogs for the shared databases
- The recovery log data sets and bootstrap data sets (BSDSs) belonging to each DB2 member

Figure 2 gives more information about the DB2 data sharing group topology. A single DB2 catalog and directory provides for a single definition of all shared DB2 objects. Changes to the definitional data can be made from any DB2 member and need to be made

Figure 1 DB2 data sharing group configuration



only once to be put into effect across the entire DB2 data sharing group.

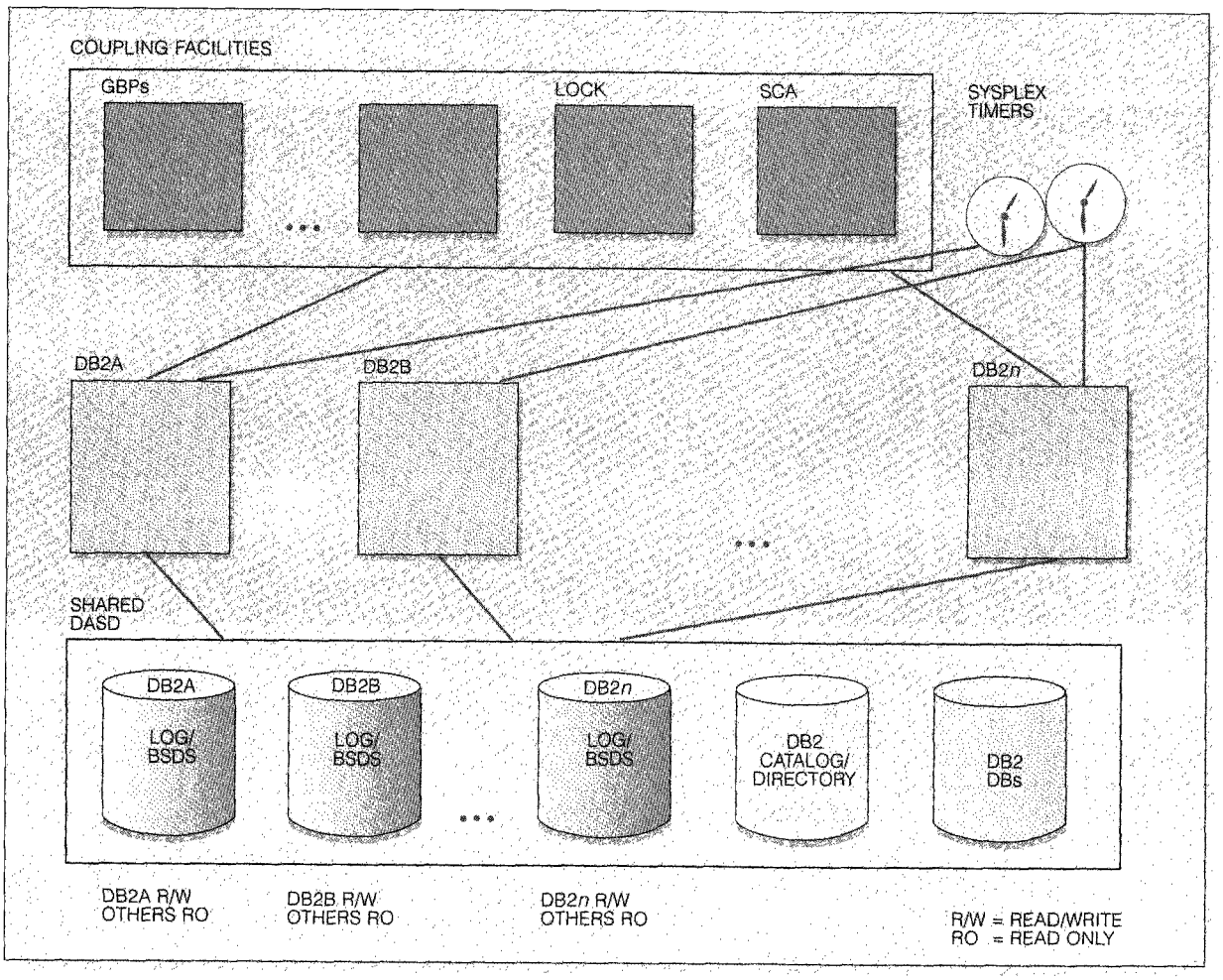
DB2 uses the following CF structures for data sharing:

- The CF lock structure for global locking
- Group buffer pools (GBPs), which are CF cache structures used for inter-DB2 buffer coherency. Up to 60 GBPs may be defined, corresponding to the 60 buffer pools that may be defined in a single DB2 member.
- The shared communications area (SCA), which is a CF list structure used for recording the exception status of databases and for supporting other internal optimizations that depend on group-wide information

The MVS central processing complexes (CPCs) are connected to the CFs with CF channels and high-speed fiber optic links that allow MVS to interact synchronously with the CFs, without task switching.

As shown in Figure 2, each DB2 member writes to its own recovery log and BSDS. However, the logs and BSDSs must reside on shared DASD so that all DB2 members have access for recovery purposes. DB2 data sharing requires a sysplex timer to give a common time source across the sysplex, so that log records

Figure 2 DB2 data sharing use of the coupling facility



can be retrieved from multiple systems, in time sequence, for recovery from media failure.

Inherent in the DB2 data sharing architecture is the ability to deliver much higher levels of capacity and availability to DB2 users, because access to the DB2 databases is no longer constrained through a single DB2 DBMS instance. Also, with the data sharing capacity, DB2 installations can now add to their DB2 systems in a horizontal, more granular fashion by nondisruptively adding new DB2 members (along with new CPCs) into the data sharing group as the need arises. And because there are multiple paths to the DB2 data, installations can choose how to best balance their DB2 workload across the data sharing group to satisfy their business needs.

To achieve maximum value from its SDa architecture, DB2 V4 is designed to provide highly efficient global locking and caching capabilities through use of the CF. To demonstrate the performance and scalability of these data sharing capabilities, the IBM Santa Teresa Laboratory has measured results for the IBM Relational Warehouse Workload (IRWW).[17] This workload consists of seven transactions of varying profiles; some are update-intensive while others are read-intensive. There are seven tables that vary in size and update intensity. All the tables are actively shared for both reading and writing when running this workload on multiple DB2 members. The measured results show 13.29 percent data sharing overhead for the workload in two-way DB2 sharing (two DB2 members, each running on its own CPC), and

13.55 percent data sharing overhead in three-way DB2 sharing (three DB2 members, each running on its own CPC).[18]

The term "overhead" here means the additional CPU (central processing unit) capacity that is needed for each DB2 member in the data sharing group to provide equivalent throughput as the same number of DB2s would provide in a nonshared environment. (E.g., if two nonsharing DB2 DBMS instances could in aggregate deliver 200 transactions per second using n units of CPU capacity each, then assuming a 15 percent "data sharing overhead," if these two DB2s were coupled together for data sharing, the two DB2s in aggregate would deliver 170 transactions per second using the same n units of CPU capacity each.) Note that as the number of sharing DB2 members increases from two to three, the data sharing overhead increases almost linearly; that is, an initial "enabling" cost is incurred in moving from a nonshared configuration to a two-way shared configuration (13.29 percent in the case of IRWW), but little or no additional overhead is incurred (0.26 percent in the case of IRWW) when increasing from two-way to three-way sharing.

In the rest of this paper we explore some of the challenges encountered in extending the single-system architecture of DB2 to handle concurrent multisystem read and write access to the DB2 databases, and how these challenges were met through use of the S/390 Parallel Sysplex coupling technology.

Global locking

To support SDi or SDa, a global lock manager is required. Examples of global lock managers are the Amoeba lock manager developed at the IBM Almaden Research Center[6] and the VAXcluster** lock manager developed by DEC.[8,9]

Figure 3 shows a logical representation of the DB2 data sharing global locking structure. In this figure, one sysplex is shown that contains n MVS systems (MVS1, MVS2, . . . , MVSn) and one CF lock structure. Also, there is a DB2 data sharing member configured on each of the n MVS systems (DB2A runs on MVS1, DB2B runs on MVS2, and so on). Each DB2 member is associated with its own internal resource lock manager (IRLM). Each IRLM can be viewed as a local lock manager (LLM) that can autonomously provide intra-DB2 locking ("local locking") functions. Each IRLM, in turn, may communicate via the MVS cross-system extended services (XES) component to the CF lock structure when inter-DB2 locking ("global locking") functions are required. XES and the CF lock structure can be viewed as the global lock manager (GLM) that tracks resources locked at each LLM. At the GLM, locks are owned by LLMs, whereas at the LLMs, locks are owned by transactions. If the GLM detects lock conflict between LLMs, then the MVS cross-system coupling facility (XCF) component is used to communicate between systems to resolve the conflict. XCF can be configured to use channel-to-channel (CTC) connections for signaling or to use CF list structures (or a combination of the two).

The CF lock structure is subdivided into two parts:

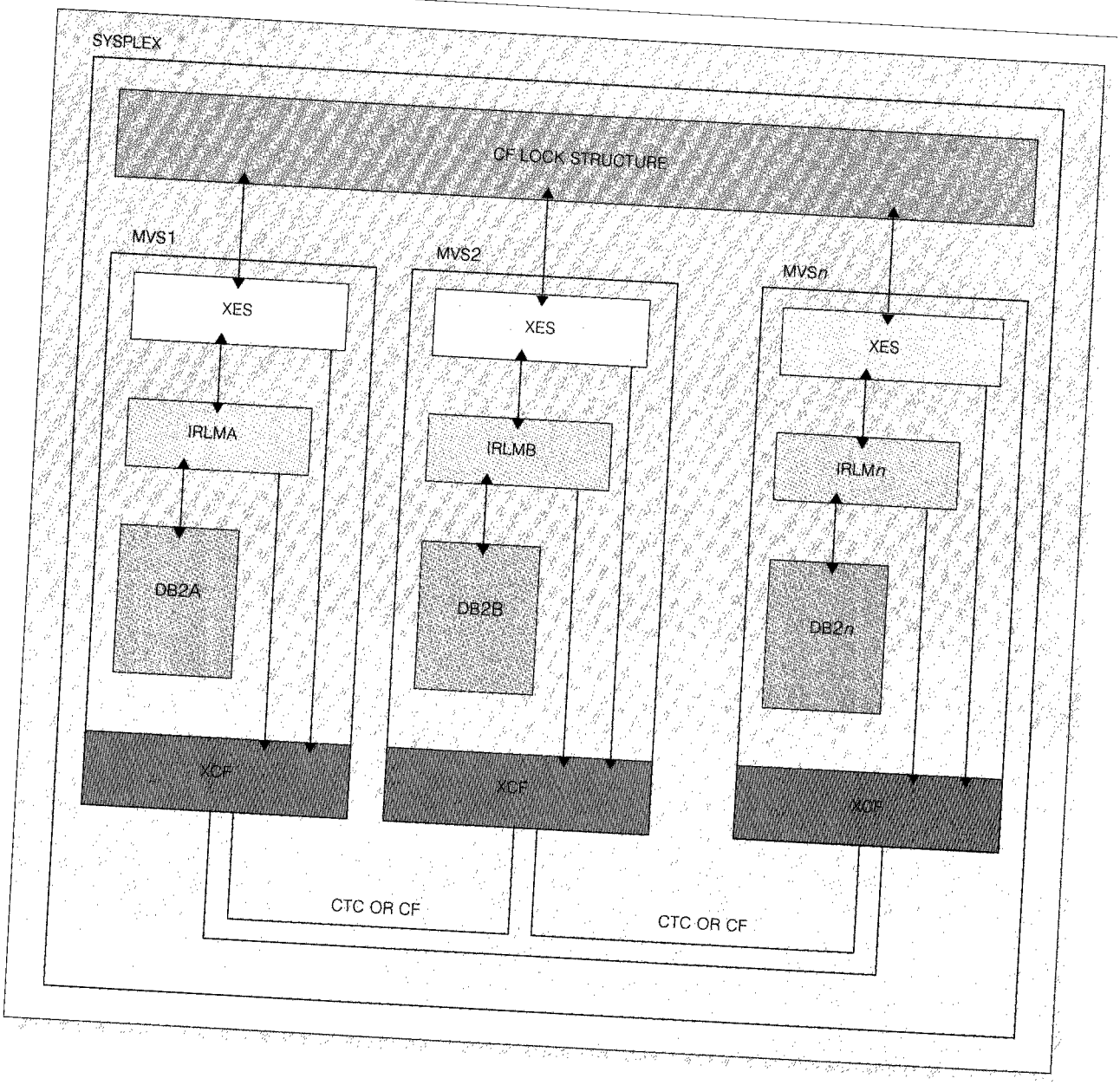
1. Lock table: used to quickly detect possible inter-DB2 lock conflict
2. Record list: used to keep track of modify locks and retained locks

When DB2 must interact with the CF lock structure for global locking, this interaction occurs synchronously, without having to suspend and resume the task. The use of the CF for global locking is one of the key distinctions between SDa and SDi. With SDi, the DBMS instances must use intersystem message passing to do global locking. With message passing, whenever a transaction requests a global lock, the transaction is suspended so that the "lock" message and the "acknowledgment" message can be sent and received. The time it takes to send these messages is measured in milliseconds (e.g., 20 msec). With SDa, in contrast, a global lock can be granted through a synchronous interaction with the CF. This CF interaction is measured in microseconds (e.g., 100 μsec).[19] It is only when inter-DB2 lock conflict is detected in the CF lock table that intersystem messaging must be used.

The premise is that all DB2 data are *shareable*. Thus any lock that is taken by a transaction on a database resource (table space, table, page, row, etc.) is a *global lock* because the database resource on which that lock is held has the potential of being accessed from multiple DB2 members. A global lock is one that provides intra-DB2 and inter-DB2 concurrency control. In contrast, a *local lock* provides only intra-DB2 concurrency control. In data sharing, almost all locks are global locks.

When a global lock is requested (e.g., on a database page), DB2 interfaces with its associated LLM (IRLM), as it would normally do in a single-system environment. IRLM then checks its local structures to de-

Figure 3 DB2 data sharing global locking configuration



termine whether or not the lock is locally grantable; that is, whether or not the lock request can be granted based on the LLM's local knowledge of the locking information. If the lock is locally grantable, then IRLM again checks its local structures to determine if the lock needs to be *propagated* to the GLM for inter-DB2 lock compatibility checking. Not all global locks need to be propagated to the GLM. This is a key performance optimization in DB2's SDa design and is explained in the next section.

Global locking optimizations. If every global lock request were to be propagated beyond the LLM, there would usually be a significant performance degradation to the system. Therefore, one of the major design goals of the DB2 locking scheme is to min-

imize the number of times that the GLM must be notified about a global lock request. This has been accomplished through two main design thrusts:

1. Suppressing the propagation of global locks to the GLM, unless it is necessary for:
   - Explicit hierarchical locking (EHL)
   - Propagating only the most restrictive state per resource per LLM
2. Lock avoidance for:
   - Type 2 indexes
   - Support for the uncommitted read isolation level

These optimizations are explained in the next four sections.

Explicit hierarchical locking. The explicit hierarchical locking (EHL) optimization implies the following: based on the current inter-DB2 locking interest for a resource, the GLM may tell an LLM that the LLM can grant locks locally on resources lower in the hierarchy (than to the requested resource). When the GLM first detects that multiple LLMs hold a lock on a resource that is higher in the hierarchy, the GLM notifies the affected LLMs that they should (1) propagate to the GLM locks that are currently held on resources lower in the hierarchy that now have the potential for intersystem lock conflict, and (2) start propagating to the GLM new lock requests on the resources.[20]

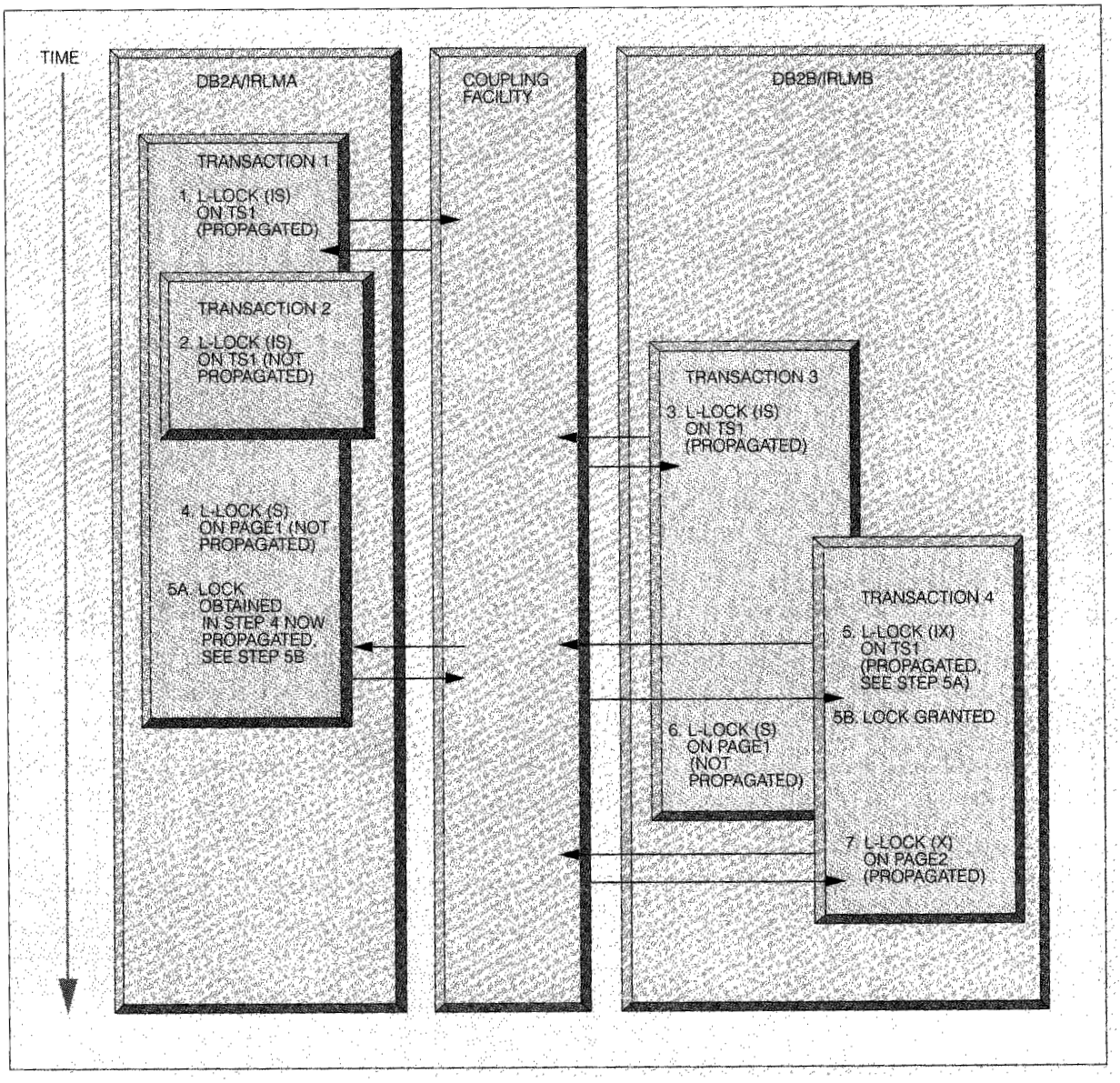
DB2 data sharing implements the EHL optimization by converting the current DB2 implicit lock hierarchy (the lock manager is not aware of the hierarchy) to an explicit lock hierarchy (the lock manager now becomes aware of the hierarchy). DB2 has always locked database objects in an implicitly hierarchical fashion. That is, gross-level, or "parent" locks are obtained first (table space, partition, table), usually in intent-share (IS) or intent-exclusive (IX) states, and then the locks on the lower level, finer granularity database resources (page, row), are subsequently obtained as they are read or updated. The implicit hierarchical locking scheme has allowed DB2 to (1) support lock escalation, and (2) allow users to have an option to specify the locking granularity, i.e., the table space, table, page, or row level.

With EHL, using page locking in a simple table space as an example (the same concepts apply for row locking and for partitioned or segmented table spaces), the locking protocol is to first lock the parent resource (table space) and remember the IRLM lock "token" that is associated with that parent resource. Then, as the "children" (pages) are locked, DB2 passes the parent lock token associated with the table space to which the page belongs so that the association of the child to the parent is made known to IRLM. By explicitly knowing the hierarchical relationship between parent and child, IRLM, working with the GLM, can dynamically determine whether there is inter-DB2 interest on the parent and then propagate or not propagate the lock requests on the children to the GLM accordingly.

Figure 4 shows an example. Here are two DB2 data sharing members, DB2A and DB2B, and transactions executing in each DB2 member that are accessing pages in the same table space (TS1). Each DB2 member communicates with its own LLM (IRLMA and IRLMB). For simplicity, XES is left out of this picture, and the GLM is represented by the CF. The following events happen in time-sequence order:

1. On DB2A, Transaction 1 gets an IS lock on TS1. Because this is the first lock on TS1 from DB2A and because the table space is the highest level in the DB2 lock hierarchy, this IS lock on TS1 is propagated by IRLMA to the GLM.
2. On DB2A, Transaction 2 gets an IS lock on TS1. Because IRLMA has already propagated an equally or more restrictive lock state to the GLM for TS1 (an IS lock has already been propagated on behalf of Transaction 1), the IS lock for Transaction 2 does not need to be propagated.
3. On DB2B, Transaction 3 gets an IS lock on TS1. Because this is the first lock on TS1 from DB2B and because the table space is the highest level in the DB2 lock hierarchy, this IS lock on TS1 is propagated by IRLMB to the GLM.
4. On DB2A, Transaction 1 gets a share (S) lock on Page 1. (Because Page 1 is contained in TS1, TS1 is the parent of Page 1.) Because the highest level inter-DB2 interest on the parent lock (TS1) is read-only (RO), the S lock on the child (Page 1) is granted locally by IRLMA without propagating it to the GLM. There is no chance of inter-DB2 lock contention on the children. (Only S locks can be requested on the children so far because no transaction has yet indicated an intent to update TS1; that happens in the next step.)
5. On DB2B, Transaction 4 gets an IX lock on TS1, indicating an intent to update one or more pages in TS1. Because the IX state is more restrictive than any lock state that has previously been propagated from IRLMB for TS1 (only IS has previously been propagated), this IX lock is propagated by IRLMB

Figure 4 Scenario showing lock propagation



to the GLM. Also, at this point the S lock that was acquired by DB2A on the Page 1 child in step 4 must be propagated to the GLM, because now that DB2B has established a lock on TS1 that indicates an intent to update pages belonging to TS1, there is the potential that S locks on the TS1 children (pages) from DB2A could hit contention with exclusive (X) locks on the TS1 children (pages) from DB2B.

6. On DB2B, Transaction 3 gets an S lock on Page 1. This lock does not have to be propagated to the GLM because DB2A still has RO interest in the parent (TS1), and thus any S locks on the children from DB2B still cannot possibly hit contention with S locks on those children from DB2A.
7. On DB2B, Transaction 4 gets an X lock on Page 2. This lock must be propagated by IRLMB to the GLM because DB2A has RO interest in the parent

(TS1), and so X locks on the children from DB2B have the potential of conflicting with S locks on those same children from DB2A.

When data are not *actually* inter-DB2 read/write (R/W) shared, EHL allows DB2 locking in a data sharing environment to have nearly equivalent performance as in a DB2 with no data sharing. The only added cost is that of propagating to the GLM some of the parent locks.

Propagating only the most restrictive state. At the GLM level, locks are owned by LLMs and not by transactions. Therefore, once an LLM has made a lock state on a given resource known to the GLM, subsequent locks granted by the LLM on that resource in an equal or less restrictive state do not need to be communicated to the GLM.

In the example just described, the IS lock on TS1 acquired by Transaction 2 does not need to be propagated to the GLM because an equally or more restrictive lock state has already been propagated from IRLMA for TS1.

Type 2 indexes. Type 2 indexes are a new type of index structure introduced in DB2 V4 in which there is no locking within the index; locks are acquired only on the data. This is in contrast with Type 1 indexes from previous releases of DB2 (and which are still supported in V4) in which locks are obtained not only on the data but also on the pages (or subpages) within the index. With Type 2 indexes, many of the locks that were obtained in previous releases of DB2 can now be avoided and this can significantly reduce the locking intensity of a given workload, which will in turn significantly reduce the overhead for data sharing global locking for the workload.

Uncommitted read isolation level. DB2 V4 also introduces the uncommitted read (UR) isolation level to allow applications to avoid locking and thus to read uncommitted data. Of course many applications cannot tolerate reading uncommitted data, but for those that can, the UR isolation level provides an effective way to improve concurrency and performance. And, as mentioned earlier, any avoidance of locks in a data sharing environment reduces overhead.

Modify locks and retained locks. With SDa and SDi, if one of the DBMS instances fails, then the data can still be accessed through any of the surviving DBMS instances because all of the data can be accessed by all of the DBMSs. (This is not true with SN). However, if there were transactions that were in progress and had not yet reached a point of consistency at the time of the failure, then the portions of the database that these transactions had locked for update must be protected in some way, so that the surviving DBMS instances are prevented from accessing the inconsistent data.

DB2 data sharing uses *modify locks* and *retained locks* to provide this protection. A modify lock is a lock held on a resource that is in the process of being updated, or modified. If a DB2 member fails, then all the modify locks held by the DB2 member at the time of the failure are converted into retained locks. Retained locks persist across the failure, and thus can continue to protect database resources that were in an inconsistent state at the time of the failure from being accessed by other DB2 members. Retained locks are held at the GLM level and thus are owned by the LLMs, not by transactions. Retained locks are not needed for resources that were accessed as RO at the time of the failure; these locks can be released.

Retained locks continue to be held until the failed DB2 member completes its restart recovery and brings the database resources back to a consistent state. If another DB2 member attempts to obtain a lock on a resource while there is still an incompatible retained lock on that resource, IRLM immediately rejects the request, and the user receives the message "resource unavailable." (There is an installation option available in DB2 that, if activated, causes IRLM to wait for a period of time for a retained lock to become available instead of immediately rejecting the request; this option would probably be used only when there is automation in place to automatically restart failed DB2 members, for example through use of the MVS automatic restart manager facility.)

As an example, suppose transaction TX1 on DB2A wants to update page P1 in table space TS1 and also wants to read page P2 in table space TS2. In this example, TX1 first gets the "intent-exclusive" (IX) lock on TS1 as a modify lock and the "intent-share" (IS) lock on TS2 as a nonmodify lock. Next TX1 attempts to read P2 and gets a share (S) lock on P2 as a nonmodify lock. Finally, TX1 decides to update P1 and gets the exclusive (X) lock on P1 as a modify lock. Now, before TX1 commits, suppose that DB2A fails. Then, the modify locks that were held at the time of the failure are converted to retained locks to persist beyond the life of DB2A. So the IX modify lock on TS1 and the X modify lock on page P1 are converted to retained locks. (The IS lock on TS2 and the

S lock on page P2 are not converted to retained because these locks were held for RO purposes; these locks are released when DB2A fails.)

To continue with the example, suppose transaction TX2 on DB2B wants to do the same thing that TX1 did. TX2 first requests the IX lock on TS1 as a modify lock, and IRLM grants the lock even though there is a retained lock on TS1 because the requested state (IX) is compatible with the retained state (IX). Next TX2 requests the IS lock on TS2 as a nonmodify lock and IRLM grants the lock. (There is no retained lock on TS2.) Next TX2 attempts to read P2 and gets a share (S) lock on P2 as a nonmodify lock, and again this is granted. (There is no retained lock on P2.) Finally TX2 is ready to update P1 and requests the exclusive (X) lock on P1 as a modify lock. But this lock is rejected because it is incompatible with the retained X lock that is held on P1 by the failed DB2A. (Note that even if TX2 had wanted only an S lock on P1, this too would have been rejected by IRLM because an S lock is also incompatible with the retained X lock.)

Once the retained locks are purged, other DB2 members can again read and update these database resources.

To complete the example, now DB2A restarts, and as part of the restart recovery process, the incomplete work that had been done by TX1 prior to the failure is backed out. After the restart recovery process has brought the database resources that DB2A was working on back to a consistent state, DB2A "purges" the retained locks that are still held on its behalf. Once the retained locks are purged, other DB2 members can once again freely read and update these database resources according to the normal rules on intertransaction locking.

Modify locks and retained locks are kept in the *record list* portion of the CF lock structure. Each IRLM also keeps a local copy of all the retained locks for fast reference. The redundancy in tracking the retained locks is an important availability consideration. Because of this redundancy, the retained locks can survive a failure of the CF or of all the IRLMs in the group.

When DB2 requests a modify lock, IRLM, through MVS XES services, must interact with the CF lock structure for two distinct operations:

1. The lock table must be consulted for inter-DB2 lock compatibility checking.
2. The record list must be updated to track the modify lock on a resource.

These two operations are bundled in one call to the CF and occur synchronously to the requesting task. An interaction with the CF for a modify lock is slightly more expensive than an interaction for a nonmodify lock; a nonmodify lock does not need an entry in the record list.

Intersystem buffer coherency

We begin this section with an overview of DB2 data buffering without data sharing. DB2 uses in-memory database buffering to minimize physical I/O activity between the CPC and DASD. A cached database page is concurrently referenced or serially updated by multiple transactions within a DB2 subsystem (i.e., DB2 caches database pages beyond transaction usage). Currently, DB2 supports 50 buffer pools of 4K (thousand) page size buffers and ten buffer pools of 32K page size buffers. A 4K page size buffer pool supports data access for 4K page size page sets. A *page set* is a synonym for a table space or an index space, if no distinction is required between them.

Each buffer pool is subdivided into two levels. The first level is the *virtual buffer pool*, which is allocated from DB2's address space (i.e., the buffer space in virtual storage is backed by central, expanded, or auxiliary storage). All database references and updates are performed against buffers in the virtual buffer pools. The second level is the *hiperpool*, which is backed only by expanded memory. The hiperpool is optional and, if defined, is internally mapped to one or more DB2-owned expanded-storage-only hiperspaces. Hiperpools are only used to cache moderately referenced clean pages. To prevent double buffering of database pages, a cached database page can reside either in the virtual buffer pool or in its corresponding hiperpool, not both.

DB2 applies deferred-write logic to updated pages and does not write updated pages to disk at commit time. (Only logs are forced to the log data sets at

commit time.) This "no force at commit" policy provides significant performance advantages for transaction response time and concurrency. It also improves DASD and CPU efficiency by batching together multiple updates and multiple pages on each disk write operation.

An overview of DB2 data buffering with data sharing. With SDi or SDa, because there are multiple DBMS instances all with equal access to the shared databases, a single page may be cached in multiple DBMS buffer pools. Assume that page locking is used. (For DB2, the page size is equal to the block size, where a block is a unit read from or written to disk.) The locking protocol to read or write a page is: acquire a share (S) lock to read the page, and acquire an exclusive (X) lock to update the page. This protocol implies that there can be multiple readers or a single updater of the page within a DB2 data sharing group.

To provide transactional semantics, the X locks that are obtained on the updated pages are held until the transaction reaches a point of consistency (until the transaction either commits or rolls back). With page-level locking in SDi or SDa, because of the global locking mechanism that has already been discussed, we do not have to be concerned with the intersystem buffer coherency problem as long as the transaction locks remain held; the X locks that are held by the transaction on the updated pages prevent the other sharing DBMS instances from updating or referencing those same pages under locks. But as soon as the transaction reaches a point of consistency and releases its X locks on the pages that it has updated, a different transaction that is executing on a second DBMS instance can obtain the locks on those same pages and can manipulate them in the local buffer pool of its corresponding DBMS instance. And, if proper controls are not in place, the cache coherency problem can be readily visualized—a down-level version of the page (a version of the page that does not reflect the latest committed updates to the data) might be read into the local buffer pool from external storage (e.g., disk) or previously cached in the local buffer pool of the second DBMS instance and used as-is.

To prevent these problems, SDi or SDa systems must implement some form of intersystem cache coherency protocols. DB2 data sharing does this by using a *force-at-commit* policy for updated database pages. Force-at-commit implies the following:

1. The updated page has to be written to external storage (DB2 uses the CF) so that other DB2 members can read the latest version.
2. The now down-level versions of the page that are cached in other DB2 buffer pools have to be cross-invalidated (XIed).

In the other DB2 members, any subsequent access to the XIed buffer pool page needs to detect the invalid condition of the page and to refresh the current version of the page from external storage.

In SDi, the buffer coherency task is accomplished by using disk storage to externalize the updated pages, and by using intersystem message passing to send the XI signals. When the transaction reaches a point of consistency, the DBMS must use I/O protocols to write each updated page to disk, and it must also send messages to (and receive acknowledgments from) each peer DBMS to ensure that the updated pages are cross-invalidated. When another DBMS detects that a buffered page has been cross-invalidated, it must use I/O protocols to refresh the page from disk. Because of the long latencies involved with the disk I/O and message-passing operations (on the order of several milliseconds for each disk or message-passing interaction), the performance penalties for maintaining buffer coherency in SDi can be severe.

In SDa, CF cache structures are used, instead of disk storage and message passing, to maintain intersystem buffer coherency when the force-at-commit protocol must be used.[21] In the DB2 implementation, we call a CF cache structure a *group buffer pool*, or GBP. A CF cache structure is an intelligent external store that handles both caching the pages and sending the XI signals. In SDa, a DBMS can use the high-speed CF channels and fiber optic links to write each updated page to the CF. The CF can actually store the page in its central storage and then send the XI signals. The time it takes to write (or read) a 4K page to (or from) the CF cache structure is measured in microseconds (e.g., 175 μsec).[19] The XI signals are processed by the CF channel hardware on the receiving systems without causing any processor interrupts. When a DBMS instance detects that a locally buffered page has been cross-invalidated, it can usually refresh the page from a CF cache structure very quickly and avoid invoking I/O protocols to retrieve the page from disk.

The use of the CF cache structures for intersystem buffer coherency is another distinguishing factor of SDa from SDi. Because CF interactions are much faster

than disk and message-passing interactions, the read and write operations to the CF can be done synchronously to the program logic, without the task-switching overhead that is necessary to deal with the long latencies of disk I/O and message passing.

For good performance, DB2 uses the CF cache structures (GBPs) as "store-in" caches, such that the version of the page in a GBP can be more recent than the one on disk. For example, when force-at-commit is applied, the updated page is written to the GBP so that the latest version of the page resides there, and the version of the page that resides on disk is now down-level.

In data sharing, DB2 data continues to be cached in each DB2 member's local buffer pools. All references and updates to DB2 pages continue to be done through the virtual buffer pools. The GBPs are used only to maintain inter-DB2 buffer coherency, and cannot be directly referenced by DB2 application programs. The buffer manager (BM) component of DB2 automatically manages the caching of the data in the buffer pools; application programs are not aware that a GBP may be in use.

The sections that follow give more detail about the way GBPs are used to maintain inter-DB2 buffer coherency.

Coherency protocol to read a page. Figure 5 shows a comparison of a simple transaction flow for reading a page between a single-system and a data sharing environment. The first difference we see is that the page lock must be global for data sharing, as explained earlier. The second difference is that when the BM finds the page cached in the local buffer pool, before the page can be used it must first be checked to see if it is still valid. (It might have been cross-invalidated.) If the page is valid, the only extra overhead that was incurred for data sharing was the extra expense of the global lock (if it was propagated to the GLM) and of checking the validity of the page.

If the page was found to be invalid in the local buffer pool, then the extra data sharing overhead of refreshing the page from external storage might have been incurred. But usually the page can be refreshed from the GBP, which is a relatively quick operation (e.g., 175 μsec for a 4K page), without having to go to disk.

To check page validity, the BM consults a bit array in the hardware storage area (HSA) of the CPC. This bit array is also referred to as a "local cache vector." In the bit array, one bit is associated with each buffer pool (virtual pool plus hiperpool) page frame indicating whether or not the associated cached page is valid or invalid.

Looking again at the flowchart in Figure 5, if the requested page is not found in the local buffer pool, then in the single system the BM would read the page from disk. In data sharing, before going to disk to get the page, the BM first issues a "read-and-register" (RAR) request to the CF to attempt to read the page from the GBP, and to register the page for XI (the read and the register are bundled in one CF request):

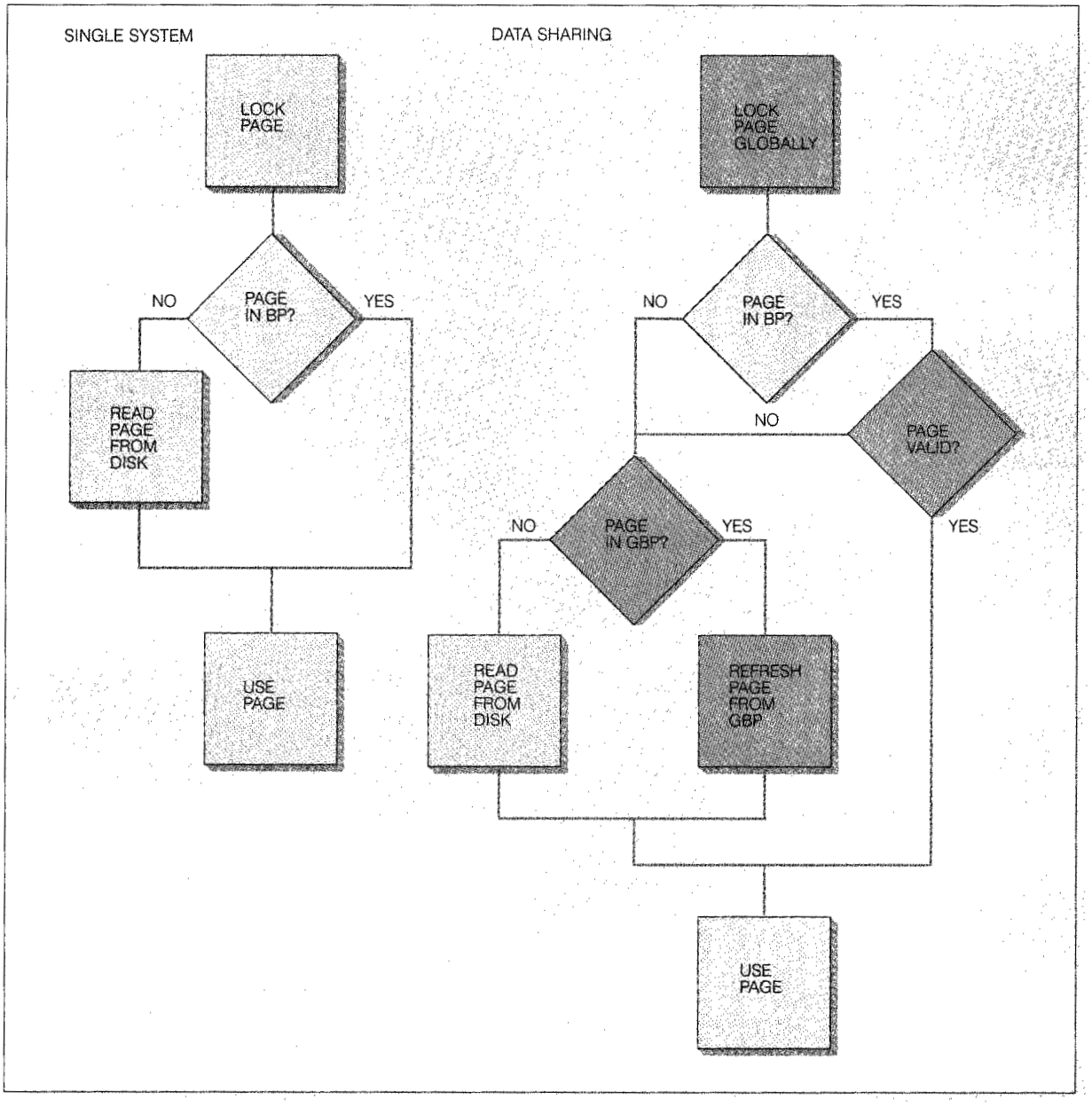
- If the page is found in the GBP, it is returned, and the page is registered for XI. If this happens, the data sharing case actually performs better than the single-system case because the page is refreshed from the CF much faster than it would be from disk. This raises the interesting possibility of using the CF as a fast intersystem cache. DB2 provides an option that can be activated at the page set or partition level to cache clean data in the GBP so that data have a greater chance of being able to be read in from the CF instead of from disk. This option is explained further in a later section.
- If the page is not found in the GBP, it is still registered for XI, but the BM must read the page from disk. If this happens (which is the more likely case), then the extra data sharing overhead is the extra call to the CF to register the page for XI, which is not necessary in the single system.

Coherency protocol to update a page. A typical transaction flow for updating a page with data sharing, assuming page-level locking, follows:

- The transaction globally locks the page in X mode.
- The transaction updates the page in the buffer pool.
- Prior to releasing the X lock at commit time, the BM issues a CF request to write the updated page to the GBP. The CF in turn sends the XI signals to the other systems where the page is cached. A CF channel on each receiving system processes the XI signal, without causing a processor interrupt, by flipping the corresponding bit in the bit array in the HSA to "invalid." The page is written to the GBP as "changed," and now the version of the page on disk is down-level with respect to the one stored in the GBP.

The data sharing coherency overhead associated with the update transaction is the writing of the updated

Figure 5 DB2 data sharing read protocol



pages with the force-at-commit policy, which increases the transaction path length, thus marginally increasing transaction response time and lock hold time. The CF write request for a 4K page should normally occur synchronously to the transaction (no task switching).

Castout. A GBP can be viewed as an extension of the DB2 members' associated local buffer pool. A GBP is used for intersystem buffer coherency; it is not used as permanent storage. When a changed page is written to a GBP, it must eventually be written to permanent storage on DASD. The process of writing the

changed pages from a GBP to DASD is called "castout."

Because there is no direct link from a CF to DASD, the DB2 castout process must read the changed page into processor storage, and then write the page from processor storage out to DASD. The castout of a page consists of the following logical steps:

1. A DB2 reads the page for castout. A castout (CO) indicator is set on in the CF hardware for this page. Its purpose is to prevent more than one DB2 from attempting to castout the same page. The CO indicator does not block transactions from accessing the page in the GBP for read or write purposes. When a DB2 member reads a page for castout, the page is read into a DB2 private storage buffer (not into the buffer pool).

2. The DB2 writes the page to disk. The write commands are batched to include several pages.

3. The DB2 resets the CO indicator. Normally, at this time the page is marked as "clean" in the GBP. (If a new updated version of the page has been written to the GBP while the CO indicator was set on, then when the CO indicator is reset, the page remains marked as changed in the GBP.)

   It is important to note that once a page is marked as clean it remains cached in the GBP. This gives the performance advantage of refreshing pages into the local buffer pools from the CF instead of from disk.

   A page marked as clean in the GBP becomes a candidate to be "stolen" (reclaimed), using a "least-recently used" methodology, so that the CF storage can be reassigned as new pages are written to the GBP.

   Just as the castout write commands are batched to minimize disk interactions, so too are the resets of the castout indicators to minimize CF interactions.

DB2 uses a robust algorithm for castout that has the following important characteristics:

- *Nonblocking:* Transactions are not prevented from read and write access to the data while they are being processed for castout.
- *Fault tolerant:* Failure of the DB2 that is casting out does not disable the castout activity. Instead, it is automatically taken over by some other DB2 in the data sharing group.
- *Distributed load:* No single DB2 in the data sharing group is burdened with the entire castout load. Instead, the work is shared by the DB2s.

The castout work is performed by "castout engines," which run as MVS system request blocks (system dispatchable units of work) in the DBM1 address space. Castout "ownership" is automatically assigned to DB2 members on a page set or partition basis. (The term "page set" will be used throughout the remainder of this paper to generically refer to a page set or a partition of a partitioned page set, unless a distinction needs to be made.) The first DB2 to update the page set becomes the castout owner for that page set. Subsequent DB2s that update the page set become the "backup" owners. A backup owner may assume castout responsibility for the page set if the original owner releases its R/W interest, or if the original owner should fail. (If a DB2 member fails, MVS automatically cleans up any castout indicators that have been set for that member.)

Castout is scheduled based on changed-page thresholds. This is similar to the way DB2 schedules writes of changed database pages from the buffer pool to disk today. There are two castout thresholds that DB2 monitors:

- Castout class threshold
- GBP threshold

Castout class threshold. In each GBP, the CF manages a fixed number of castout class queues. (The current CF models support a maximum of 1024 castout class queues and DB2 requests the maximum.) Whenever DB2 writes an updated page to the GBP, it must specify the castout class queue to which the page belongs. DB2 internally maps the updated pages that belong to the same page set to the same castout class queue, using a hashing algorithm to assign a castout queue number to a page set. Due to the limited number of castout class queues, it is possible that more than one page set gets mapped into the same castout class queue.

When the CF write operation for a page completes, as part of the feedback, the CF indicates how many changed pages are in the associated castout class queue. If this number exceeds the "castout class threshold" value, then the DB2 member that is the

castout owner of the page set is notified to initiate castout operations for the page set.

The castout class threshold default is 10 percent. (Once the number of pages in the castout class queue reaches 10 percent of the total number of pages in the GBP, then the threshold is reached.) The installation can dynamically alter this value via an operator command.

GBP threshold. DB2's castout algorithm uses a two-level logical hierarchy of DB2 members that communicate via messages. At the lower level of the hierarchy are DB2 members in charge of castout for page sets, and at the upper level is the DB2 member in charge of castout for the whole GBP structure. The DB2 at the higher level accomplishes its work by calling upon services of DB2s at the lower level. This two-level hierarchy ensures that castout will be done even if thresholds are not being reached on the castout class queues. (E.g., maybe the updates are evenly distributed among several castout class queues and no single castout class has reached the threshold.) The two-level hierarchy also ensures that castout processing occurs for page sets that might not have a castout owner (e.g., due to one or more DB2 member failures).

The DB2 at the higher level of this castout hierarchy is called the GBP *structure castout owner.* One DB2 member establishes itself as the structure castout owner for a given GBP, and the other DB2 members interacting with that GBP establish themselves as backup owners for the structure. One of the backup owners will assume the ownership if the original owner terminates normally (e.g., the DB2 member is stopped via an operator command) or abnormally (e.g., the DB2 member fails).

On the completion of a timer interval, the structure castout owner wakes up and queries the number of changed pages in the GBP (with one CF interaction). If the number of changed pages exceeds the GBP threshold value, then the structure owner notifies a subset of the page set castout owners on a round-robin basis to castout enough pages to get back down to a reverse threshold. (Normally, the reverse threshold is 10 percent below the GBP threshold. E.g., if the GBP threshold is 50 percent, then the reverse threshold is 40 percent.)

The GBP threshold default is 50 percent. (Once the number of changed pages in the GBP reaches 50 percent of the total number of pages in the GBP, then the threshold is reached.) The installation can dynamically alter this value via an operator command.

Reducing the overheads associated with buffer coherency. A key design goal of DB2 data sharing is that overheads should be incurred only when there is actual inter-DB2 R/W sharing of the data. If there is no such sharing, then transactions should require little or no added path length for data sharing protocols. The explicit hierarchical locking optimization allows us to achieve this objective with respect to global locking. But for inter-DB2 buffer coherency protocols, we need a different mechanism, because the caching of data in the buffer pool is not directly related to transaction locking. Said another way, the "physical" (device-oriented) interactions that take place at the buffer manager level are not directly related to the "logical" (transaction-oriented) interactions that take place at the data manager level. For example:

- A transaction may hold an IX lock on a table space without ever actually updating any page belonging to that table space. (This may be especially common when the plan[22] is bound with ACQUIRE(ALLOCATE) or in thread reuse situations where the plan is bound with RELEASE(DEALLOCATE).)
- Transaction locks are not obtained on index page sets, so the cache coherency controls for index pages could not be directly tied to transaction locks held on the indexes (index coherency controls would need to be tied to the locks held on the associated table spaces, which may overstate the actual intersystem read or write activity on the index).
- Pages belonging to a page set may remain cached in the buffer pool long after transaction locks have been released.

In this section we describe ways that buffer coherency overhead can be avoided or reduced by the BM being aware of the inter-DB2 physical read or write activity on a page set. The key observation is that overhead for page set coherency should exist only if there is at least one updating DB2 *and* other DB2s are using that page set. This may seem obvious; however, the challenge is for a DBMS to react appropriately to the dynamics of intersystem interest changes. We explain why dynamic adjustment of interest is worth the trouble, rather than always incurring the overhead required for the most conservative case.

Table 1 Page set physical interest level dynamics

From	To	When
None	RO	Physical open
RO	R/W	Pseudo open
R/W	RO	Pseudo close
Any	None	Physical close

The mechanism by which the BM dynamically tracks the inter-DB2 interest is called "physical locking" (P-locking). Physical locks (P-locks) are acquired by the BM on the page set to declare the physical read or write interest of a DB2 member in the page set. The physical read or write interest reflects the actual physical access characteristics of the transactions that are running on that DB2 member; this is in contrast to the logical interest of the transactions, which is manifested in the transaction locks ("logical locks," or L-locks) that reflect the logical data access intent of the transactions. When dealing with buffer coherency issues, it is the actual physical read or update operations against the data that we are concerned with, not the logical intent to read or update the data. Thus the DB2 data sharing buffer coherency protocols are controlled by P-locks, not by L-locks.

Table 1 shows how the BM component for each DB2 member declares its physical interest level in a given page set. As the table indicates, a DB2 member can have any one of three levels of physical interest in a given page set:

- No interest. The DB2 member does not have the page set physically open, and thus is not accessing the page set for read or update operations. No P-lock is held on the page set.
- RO interest. The DB2 member has the page set open for RO access, and thus transactions running on this member are reading data belonging to the page set, but not updating any of the data. The P-lock on the page set is held in RO mode.
- R/W interest. The DB2 member has the page set open for R/W access, and thus transactions running on this member may be reading or updating data belonging to this page set. The P-lock on the page set is held in R/W mode.

The term *pseudo open* refers to the point in time when the page set is first physically updated by any transaction running on a DB2 member after the page set was previously in an RO state. (I.e., at pseudo open, the BM converts the DB2 member's physical interest on the page set from RO to R/W status.) Conversely, the term *pseudo close* refers to the point in time when the BM determines that the page set has not been updated recently and converts the DB2 member's physical interest on the page set from R/W state back to RO state. (In DB2, pseudo close is controlled by the PCLOSET and PCLOSEN installation parameters.) By converting the member's interest back to an RO state, the BM can narrow the ranges of log records that must be scanned for media recovery or restart recovery for the page set.

It is important to note that P-locks are fundamentally different from the transaction locks (L-locks) discussed earlier. The purpose of P-locks is to ensure that the proper cache coherency protocols are used in a multisystem environment; P-locks have no meaning in a single-system environment. In contrast, the purpose of L-locks is to control intertransaction logical consistency for concurrent data access; L-locks have meaning both in a single-system and a multisystem environment. P-locks are owned by the DB2 member, while L-locks are owned by the transaction.

Another key difference between P-locks and L-locks is that P-locks are *negotiable*. That is, if one of the DB2 members changes the state of its P-lock on a resource (page set or partition) due to a change in the physical access characteristics on the resource (e.g., the DB2 member is going from RO to R/W physical interest on a page set) the other DB2 members that hold a P-lock on that resource will be notified by their respective LLMs of this change in the inter-DB2 interest on the P-lock, and each DB2 member can then dynamically make any necessary adjustments in the cache coherency processing for the resource and then downgrade or upgrade its P-lock state (negotiate the P-lock) accordingly. The P-lock negotiation process allows DB2 to react dynamically to the changes of inter-DB2 interest and to enact intersystem buffer coherency protocols only when there is *actual* physical inter-DB2 R/W sharing of a page set.

Table 2 summarizes the DB2 data sharing buffer coherency protocols that are enacted by a DB2 member for each of the five different physical "access levels" that the member may have for a particular page set. A DB2 member's access level for a page set depends on two factors: (1) the member's physical interest level in the page set (RO or R/W), and (2) the aggregate of the interest levels of the other DB2 members (RO, R/W, or none). Using the P-locking mechanism just described, a DB2 member can dynamically

Table 2 Cache coherency protocols for the page set access levels

Interest		Implies for This DB2			
This DB2	Other DB2s	Access Level	Force at Commit	Check Page Validity on BP Hit	Go to GBP on BP Miss
RO	None or RO	1	—	No	No
RO	R/W	2	—	Yes	Yes
R/W	None	3	No	No	No
R/W	RO	4	Yes	No	Yes
R/W	R/W	5	Yes	Yes	Yes

adjust between access levels as the physical data access characteristics of the workload vary over time.

If its access level on a page set is 2, 4, or 5, then the member must use the GBP to maintain buffer coherency for that page set. In these cases we say that for this member the page set is *group-buffer-pool-dependent* (GBP-dependent). If its access level on a page set is 1 or 3, then the DB2 member does not need to use the GBP to maintain buffer coherency for the page set. In these cases we say that the page set is non-GBP-dependent. Non-GBP-dependent page sets incur only the insignificant overhead needed to manipulate the page set P-lock during page set open and close events.

Note also that a DB2 member with a page set access level of 4 can avoid the page validity check when it finds a page belonging to that page set in its local BP. In this case the member can take advantage of its knowledge that all other DB2 members in the group have at most RO physical interest in this page set, and therefore any page belonging to it that this member finds in its local BP cannot possibly have been cross-invalidated by an update from another DB2.

Handling subpage concurrency. For DB2, the page size is the same as the block size, where a block is a unit of disk (or CF) that is read or written. Therefore, when page-level transaction locking is in effect with DB2 data sharing, DB2 can ensure that physical consistency[23] of the pages is maintained, because the global page lock ensures that only one transaction process can update the page at a given time. However, when subpage concurrency is allowed (e.g., row-level locking on data pages) the global transaction lock is not held at the block level. Unless prevented, transactions running on different DB2 members could be allowed to update the same block (page) at the same time, with one DB2 member possibly backing out the updates of another as the block gets written to external storage.

In a single-system environment, DB2 obtains page latches to ensure that physical consistency of the pages is maintained; that is, only one transaction can move the bits around on the page at a time, and readers are not permitted to look at the page until the updater has finished. DB2 supports exclusive (X) and share (S) modes for its page latches.

A latch is a memory-based serialization technique; that is, the control of which transaction holds the X latch, or which transactions hold the S latch (and which transactions are waiting for the latch) on a given resource is accomplished by using compare-and-swap logic on a specific area in virtual memory associated with that resource (usually a control block). However, in a multisystem data sharing environment, because we now have distributed memories (each DB2 member has its own virtual memory spaces), the memory-based latching technique no longer guarantees serialization across all the transactions. We need to extend the scope of the page latch across all DB2 members to maintain the physical consistency of the page when subpage concurrency is allowed.

To do this, DB2 uses a P-lock on the page as a "global page latch." The page P-lock has similar properties to the page set P-lock described earlier. The page P-lock is owned by the DB2 member (not the transaction), and the page P-lock is negotiable. Because the P-lock is not owned by any transaction, the P-lock on a page can be released before (or after) the transactions that are updating the page reach a point of consistency. Therefore, using the page P-lock, two different transactions running on two different DB2 members can obtain X locks on different rows in the same page and each transaction can update[24] its X-locked row without requiring that the other transaction reaches a point of consistency. The scenario could be as follows:

1. Transaction 1 on DB2A acquires an X L-lock on Row 1 of Page 1.
2. Transaction 2 on DB2B acquires an X L-lock on Row 2 of Page 1.
3. Transaction 1 updates Page 1 in DB2A's local BP to reflect the new contents of Row 1. Updates to pages are always done under the X page latch. Before the update is allowed, DB2A acquires the X mode P-lock on Page 1. Once the P-lock is obtained, Transaction 1 moves the bits around on Page 1 to update Row 1. The P-lock must be obtained after the page latch is already held. This order is important because it prevents the P-lock from being "stolen" from the updating transaction by another DB2 member (see step 5).
4. Transaction 2 attempts to update Page 1 in DB2B's local BP to manipulate the contents of Row 2 on the page. Before the update is allowed, DB2B requests the X mode P-lock on Page 1. However, the P-lock cannot immediately be granted because DB2A still holds it.
5. DB2A is notified by its LLM that DB2B is requesting the X mode P-lock on Page 1. DB2A responds by writing its updated copy of Page 1 to the GBP (thus cross-invalidating any locally cached copy of Page 1 that DB2B might have) and then releasing its P-lock. The writing of the page to the GBP must be done under the local page latch. Note that the P-lock can be released even though Transaction 1 still has not reached a point of consistency.
6. DB2B is granted the X mode P-lock on Page 1. Under the P-lock on the page and the local page latch, DB2B reads Page 1 into its local BP from the GBP. This version of Page 1 reflects Transaction 1's updates to Row 1.
7. Transaction 2 moves the bits around on Page 1 to update Row 2.
8. Transaction 1 commits its update to Row 1. The force-at-commit protocol is invoked, but Page 1 is found to be "clean" in DB2A's local BP (because it had already been written in step 5), thus Page 1 is not written.
9. Transaction 2 commits its update to Row 2. The force-at-commit protocol is invoked, and Page 1 is found to be "dirty" in DB2B's local BP, thus Page 1 (containing both Transaction 1's latest update to Row 1 and Transaction 2's latest update to Row 2) is written to the GBP (and DB2A's locally cached version of Page 1, which does not reflect Transaction 2's latest update to Row 2, is cross-invalidated).

Page P-locks are not required in all cases—they are only required when accessing a GBP-dependent page set *and* subpage concurrency is allowed on the page *and* the page access is one of the following:

1. The transaction is updating the page. The page P-lock is obtained in X mode to guarantee inter-DB2 write-write serialization on the page.
2. The transaction is reading the page and must have the guarantee that it is operating against the most recent version of the page. The page P-lock is obtained in S mode. In general, this means that only those transactions with an isolation level of repeatable read[25] need to get the S mode page P-lock.

It is important to note that page P-locks are only required when the page set is GBP-dependent. If the page set is non-GBP-dependent, the page P-lock is not required. So by dynamically tracking the inter-DB2 physical interest on the page set, the added cost of page P-locking can be avoided if the page set is not physically inter-DB2 R/W shared.

It is also important to note that the P-lock is system-owned, not transaction-owned. So if a transaction needs to access a page in a manner that requires a P-lock on the page, the P-lock may already be held by that DB2 member due to a read or update to the page from a previous transaction on that DB2.

Using GBP as a global cache. Because the access speed of the CF is significantly faster than that of DASD, there may be a price-performance gain achieved by caching clean pages of data in the GBP when heavy inter-DB2 sharing is expected on a page set. Caching clean pages in the GBP can reduce the number of disk accesses by allowing pages to be read from DASD once, and thereafter be retrieved from the GBP.

The GBPCACHE clause on a CREATE/ALTER TABLESPACE/INDEX statement allows the user to specify how the GBP should be used for the page set (or partition) to support inter-DB2 sharing activity in the DB2 data sharing environment. A GBP can be used by page sets that have different sharing requirements. To facilitate this support, the GBPCACHE clause allows the specification of CHANGED or ALL.

For a GBP-dependent page set with the GBPCACHE CHANGED (this is the default) attribute, the GBP is used only for coherency, and therefore DB2 only writes updated pages to the GBP. Clean pages are

required to be registered to the GBP for cross-invalidation purposes only. Note that when registering a locally cached clean page to the GBP, the CF only has to allocate a directory entry (without the backing storage for the data) if the page is not currently registered or cached in the GBP. If a local hiperpool exists, it is used as another level cache for clean pages that belong to GBPCACHE CHANGED type page sets.

If a page set is defined with the GBPCACHE ALL attribute, in addition to writing the updated pages to the GBP, DB2 will also write the clean pages to the GBP as they are read in from DASD. In this case, the GBP is used to improve read performance as well as to maintain coherency. To prevent double caching, clean pages are not cached locally in a hiperpool, if one exists, for GBPCACHE ALL type page sets.

Logging and data recovery

When a DBMS is extended for SDi or SDa, one of the key decisions is whether or not to support a real-time merged log. For the purposes of this discussion, the merged log implies:

- The log sequence number (LSN) assigned is a monotonically increasing number across the data sharing group.
- It is possible to read the log records merged across all systems by providing their LSNs.

The choice of supporting a merged log or not affects restart recovery, media recovery, and buffer coherency schemes. When an existing DBMS is being modified for SDi or SDa, it may not be practical to support a merged log because of development and migration costs. The migration could be of both logs and data. However, if a merged log is not used, then each system will have its own log and assign its LSNs independently. If the DBMS keeps the LSN of the last update to the page in the page header[26] then the recovery algorithm assumes that the LSN is a monotonically increasing number. When LSNs are assigned independently by each DBMS instance and the page can be updated by different systems, the LSN in the page header is no longer a monotonically increasing number, and the recovery algorithm will not work correctly.

The solution to this problem is to redefine the semantics of the LSN field as the *version identifier* (id) of the page, which is a monotonically increasing number. The version id is increased each time the page is updated. For restart recovery, the version id is tracked in the log record. Therefore, the recovery algorithm would apply the following rule: if the version id in the log record is greater than the version id in the page-LSN field, then apply the log record.[27]

The advantages of this approach are that the log code, the log data sets, and the databases need not be migrated to move from the single-system to the multisystem data sharing architecture. However, the log records need to be extended.

To support the log merge capability, DB2 relies on the sysplex timer to provide a synchronized time source across systems. In the data sharing environment, the *log record sequence number* (LRSN) is a 6-byte value equal to or greater than the 8-byte time stamp value truncated to six bytes. DB2 uses the LRSN for data page versioning. A key feature of this mechanism is that the LRSNs assigned are group-wide unique without having a global counter. In the non-data-sharing environment, DB2 continues to use the log relative byte address for data page versioning.[28]

Because the LRSN is derived from the first six bytes of the time stamp, we must be careful that two consecutive updates to the same page are not made using the same LRSN value; the first six bytes of the time stamp will increment once every 16 microseconds, so if two updates are made to the same page within the same 16 microsecond interval, the second update could use the same LRSN as the first, thus not increasing the page version id from one update of the page to the next. To prevent this from happening, when generating the LRSN value for an update to a page, DB2 always passes the existing LRSN page version id to the DB2 log manager, and the log manager always ensures that the new LRSN value is higher than the previous one.

When enabling an existing DB2 subsystem for data sharing, no database migration is necessary to convert from using relative-byte-address page versioning to using LRSN page versioning. The system clock moves faster than the RBA value, so a higher page version id is always assigned.

The log component is structured with a local log manager that resides in every DB2 member and writes to its own BSDS and log data sets. These BSDSs and logs must reside on shared DASD, because whenever data recovery requires that a merged log stream be applied to the data, the log-merge function will, on behalf of the recovery process, read the DB2 member log records that are needed for the recovery and

merge them in time sequence order. The log-merge process may run on any DB2 member in the group.

To restart a DB2 subsystem, only the subsystem's log is needed; no merging is necessary.

CF-related failures

The following two types of CF-related failures may occur:

- The failure of a CF itself
- The failure of an attachment of a CPC to a CF

When a CF-related failure occurs that affects one or more DB2-managed CF structures (SCA, lock structure, or GBPs), then DB2 will take recovery actions to ensure that data integrity is maintained in the data sharing group. Of course the goal of maintaining data integrity is most important; however, the DB2 CF recovery algorithms also attempt to minimize the effect of the failure on DB2 users and applications and to return to full data sharing operations as soon as possible. While the recovery actions are in progress, there may be degraded service or even some DB2 data that are unavailable.

In order for DB2 CF recovery to work properly, a sysplex failure management (SFM) policy must be defined and activated. The SFM policy must have CONNFAIL(YES) specified, and WEIGHT values should be appropriately assigned to the systems in the sysplex. For simplicity, the discussion in this section assumes that this has been done.

CF failures in the SCA and lock structures. When the SCA or lock CF structures are lost due to a CF failure, DB2 (for the SCA) or IRLM (for the lock structure) will automatically rebuild the lost information in a new structure in the same or different CF. (The most likely scenario would be that entire CF failed, so that the information would have to be rebuilt into a different CF.) The information is rebuilt from the aggregation of information that resides in the virtual storage areas of the DB2 and IRLM members across the group. (Also, for the SCA, the BSDSs may be used to rebuild some of the lost information.)

The dynamic rebuilding of these CF structures normally takes a few tens of seconds. All access requests to these structures are suspended until the rebuild completes. Because access frequencies for the SCA are low, the disruption caused by rebuilding it is usually minimal. However, the lock structure is usually accessed relatively frequently, and transactions may experience a delay (and some lock time-outs may result) while the dynamic rebuilding of the lock structure is in progress.

If there are failed or quiescent DB2 members at the time of the CF failure, the dynamic rebuilding of the SCA or lock structure can still proceed.

If rebuilding the SCA or lock structure fails for any reason (the most common reason would be that there is no alternate CF in which to rebuild), then all DB2 and IRLM members in the group abnormally terminate, and the lost SCA or lock structure must be rebuilt from the recovery logs by the *group restart* process. Note that this implies that if only one CF is configured into the sysplex, this CF constitutes a single point of failure for the DB2 data sharing group. If two or more CFs are properly configured, then it takes a *double failure* (an initial CF failure followed by some hardware or system software failure during the rebuilding) to require a group restart.

When a CF-related failure occurs that affects its structures, DB2 takes actions to ensure data integrity.

If the entire sysplex loses power, then group restart will be needed only if the CFs in which the SCA and lock structure were allocated are not configured for nonvolatility. If the SCA and lock structures persist across the power outage, then each DB2 member can restart individually as the power is restored to each system.

CF failure in the GBPs. When a GBP structure is lost due to a CF failure, all changed data that belong to GBP-dependent page sets must be recovered from DASD and the merged DB2 logs. A key point is that the recovery proceeds from the DASD version of the data; image copies are not needed. Dynamic rebuilding, as done for the SCA and lock structures, is not feasible for the GBPs, because DB2 uses the GBP as a store-in cache, and so there is no guarantee that the changed pages that were lost in the GBP are still

in virtual memory somewhere—the changed GBP pages must be recovered from the logs.

When DB2 receives notification that a GBP has been lost, one of the DB2 members (the GBP structure castout owner) automatically initiates a process called *damage assessment* (DA). The DA process, which normally completes within a few seconds, determines which GBP-dependent page sets (or partitions) were using the failed GBP, and marks each of those objects to be in *GBP recovery pending* (GRECP) status. DB2 can complete the DA process quickly because it does not need to read the merged log in order to determine which page sets were GBP-dependent. Instead, it can quickly consult the P-lock states of the page sets using the associated BP. (The P-lock state indicates whether or not the page set is GBP-dependent, as described earlier.) This technique is especially valuable if there are failed DB2 members at the time that DA is being done; DB2 can quickly query the retained page set P-locks of the failed members to determine if any further page sets were GBP-dependent and need to be recovered.

Once DA completes, each DB2 member disconnects the GBP, which can be reallocated in the same or a different CF. DB2 does not allow a new GBP instance to be allocated while DA is in progress.

The page sets that are marked as GRECP remain unavailable for access until they are recovered. Once a page set is marked as GRECP, it is no longer GBP-dependent and can be recovered without the GBP being available. Normally the DB2 START DATABASE command is used to do the recovery. DB2 automatically determines which log ranges must be scanned from each DB2 member, and then merges those log records in time-sequence order. The number of log records that need to be scanned and merged is determined by the *GBP checkpoint*.

The purpose of the GBP checkpoint is to figure out and record the oldest log time stamp required to begin the log merge to recover lost changed pages in the GBP. The GBP checkpoint is controlled by a time interval (elapsed time) that can be set by the user via an operator command. (The default is every eight minutes.) The GBP checkpoint is performed by the GBP structure castout owner, which:

1. Initiates the castout processes, asking every page set castout owner to flush all changed pages from the castout class queues to DASD
2. Gets the restart REDO LRSN log point for each sharing DB2. This is really the oldest "write pending" for the DB2 member and may be earlier than the oldest changed page in the GBP—perhaps the change never got to the GBP, or got to the GBP after the checkpoint process completed (and before the failure occurred). The member REDO LRSN values are kept in the SCA. Each DB2 periodically updates its own REDO LRSN value. (It is normally done when the DB2 member takes its system checkpoint.)
3. Scans the GBP directory entries to obtain the earliest LRSN across all changed pages in the GBP. This is necessary because the checkpoint does not wait for the castout flush (see step 1) of the GBP to finish. The flush will normally complete by the start of the next GBP checkpoint cycle. Note that, for changed pages that are being castout, their restart REDO LRSNs (kept in the GBP) will not be moved forward until the castout DASD writes are completed. Therefore, there is no data integrity problem if DB2 initiates another GBP checkpoint before the DASD writes are completed for all castout pages that were scheduled by earlier GBP checkpoints.
4. Records the LRSNs obtained in steps 2 and 3 in the GBP checkpoint record residing in the SCA (and backed up in the BSDS). They will be used as the "GBP recover LRSNs" by the DA process when marking page sets as GRECP. These recover LRSNs are associated with the GRECP page sets and are used to determine the scan starting points in each member's log for the GBP-dependent page sets. The starting point for each member is the lower of the member's REDO LRSN value and the value of the GBP LRSN obtained in step 3.

If the entire sysplex loses power, then DA will be needed only if the CFs in which the GBPs were allocated are not configured for nonvolatility. If the GBPs persist across the power outage, then each DB2 member can restart individually as the power is restored to each system and can connect back to the GBPs without any loss of data.

DB2 Version 5 delivers significant enhancements in the area of GBP failure recovery.

CF attachment failures. Attachment failures between DB2 and the coupling facility are detected by the XES component of MVS, which notifies the affected DB2 members. Normally only one DB2 member is run per MVS system.

DB2 connectivity loss to the SCA or lock structure. When a DB2 member loses connectivity to the SCA or lock structure, there are two choices: (1) fail the affected member, or (2) dynamically rebuild the structure into another CF, which may have better connectivity. The choice depends on the importance of the work running on the MVS systems that have lost connectivity relative to the importance of the work on the MVS systems that have not lost connectivity. MVS determines the relative "importance" of the MVS systems by consulting the system WEIGHT values, as specified by the user in the active SFM policy, and the REBUILDPERCENT value for the CF structure, as specified by the user in the active coupling facility resource management policy. If the magnitude of the impact (as determined by the SFM WEIGHT values) is greater than the REBUILDPERCENT value, then DB2 or IRLM will rebuild the structure. Otherwise the affected members will terminate so that work being done on the other data sharing members can continue without being disrupted by rebuilding a CF structure.

DB2 connectivity loss to GBPs. The affected DB2 member responds to the loss of GBP connectivity by quiescing all its access to page sets dependent on that GBP, and then disconnecting the GBP. The affected DB2 remains up since it can still provide service (albeit a degraded mode of service) to access data that were not dependent on the disconnected GBP. Users running on the affected DB2 receive a "resource unavailable" condition if they try to access the affected GBP-dependent page sets until (1) the CF attachment problem is fixed, or (2) the member is stopped and restarted on another system with connectivity to the GBP, or (3) the GBP is reallocated on another CF to which this member has connectivity.

Transactions in progress at the time the GBP connectivity was lost that try to write their changed pages to the GBP receive a "no connectivity" return code. The buffer manager responds to this condition by adding the pages to the *logical page list* (LPL) for the page set. The LPL is a list of pages that are temporarily inaccessible because DB2 incurred problems when attempting to write them to external storage. The pages can be recovered using the current DASD or GBP version of the page as the base for the recovery (the image copy is not needed). If the LPL recovery fails (e.g., because the disk medium has been damaged), then the page is added to the write error page range and must be recovered using the image copy as the recovery base.

Group restart. Group restart is the process of rebuilding lost SCA or lock structure information from the DB2 recovery logs when these CF structures are lost due to CF failures and DB2 or IRLM was not able to dynamically rebuild the lost structure. Group restart should be very rarely needed if two or more CFs are configured into the sysplex. Group restart requires information from the logs of all nonquiescent DB2s in the group to rebuild the SCA or the lock structure, as follows:

- To rebuild the SCA, the current status rebuild (CSR) phase of restart must be performed by reading each member's log forward from the last complete checkpoint. No DB2 member can proceed beyond the CSR phase of restart until CSR has been performed on behalf of every member.
- To rebuild the lock structure, first the CSR phase of restart must be performed for each member, as above. Once CSR is complete for every member, then the historic status rebuild (HSR) phase of restart must be performed for every member. This phase also requires reading each member's log forward from the last complete checkpoint. No DB2 member can proceed beyond the HSR phase of restart until HSR has been performed on behalf of every member.

If both the SCA and the lock structure were lost, then CSR needs to be done only once for each member.

Group restart is initiated by restarting one or more DB2 members (with the START DB2 command). DB2 restart processing automatically determines whether or not group restart is necessary. If group restart is initiated, then DB2 automatically handles the synchronization of the various phases of restart across the members.

During group restart, all restarting DB2s rebuild the SCA or lock structure from information contained in their logs. If not all members of the group are restarted, then the started DB2s carry out group restart on behalf of the nonstarting DB2s by reading their logs. (If a DB2 member was quiesced normally at the time of the CF failure, then its logs do not need to be scanned for group restart.) Although one DB2 can perform group restart on behalf of the group, it is usually significantly faster if all of the nonquiescent members are restarted so that the group restart log scans can be performed in parallel.

Summary

The SDa architecture delivers the availability, workload balancing, and flexible growth benefits of SDi, but through use of the coupling facility SDa can avoid the high overheads of frequent disk I/O and intersystem message passing associated with SDi. Although not discussed in this paper, the architecture is also used for both interquery and intraquery parallelism.

We have shown in this paper how DB2, an industrial-strength relational DBMS for the S/390 environment, has been extended from its single-system roots to implement SDa using the CF for global locking and intersystem buffer coherency. Use of the CF is the key factor, allowing multisystem data sharing with good performance characteristics. In addition, we have described several optimizations that DB2 employs to eliminate unnecessary interaction with the CF, thus further reducing the overhead for data sharing, global locking, and buffer coherency.

We have described some of the recovery considerations for multisystem data sharing, and, specifically, DB2's implementation of retained locks, recovery logging, and CF failure recovery to ensure that data integrity is maintained across the failure of any hardware or software element in the sysplex. DB2's robust design for data sharing builds on the strengths of the S/390 Parallel Sysplex to provide DB2 users with unprecedented levels of capacity, availability, and parallelism.

Cited references and notes

1. K. Shoens, "Data Sharing vs Partitioning for Capacity and Availability," *IEEE Database Engineering* **9,** No. 1, 10–16 (March 1986).
2. M. Stonebraker, "The Case for Shared Nothing, *IEEE Database Engineering* **9,** No. 1, 4–9 (March 1986).
3. R. J. Peterson and J. P. Strickland, "Log Write-Ahead Protocols and IMS/VS Logging," *Proceedings 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, GA (March 1983), pp. 216–243.
4. J. Strickland, P. Uhrowczik, and V. Watts, "IMS/VS: An Evolving System," *IBM Systems Journal* **21,** No. 4, 490–513 (1982).
5. A. Yamashita, "Data Base Integrity at Emergency Restart in Data Sharing," IBM Invention Disclosure SA882-0110, *IBM Technical Disclosure Bulletin* **26,** No. 2, 863 (July 1983).
6. K. Shoens, I. Narang, R. Obermarck, J. Palmer, S. Silen, I. Traiger, and K. Treiber, "Amoeba Project," *Proceedings IEEE Compcon Spring '85*, San Francisco, CA (February 1985).
7. A. Joshi, "Adaptive Locking Strategies in a Multi-Node Shared Data Model Environment," *Proceedings 17th International Conference on Very Large Data Bases*, Barcelona, Spain (September 1991), pp. 181–191.
8. N. Kronenberg, H. Levy, and W. Strecker, "VAXclusters: A Closely-Coupled Distributed System," *ACM Transactions on Computer Systems* **4,** No. 2, 130–146 (May 1986).
9. T. K. Rengarajan, P. Spiro, and W. Wright, "High Availability Mechanisms of VAX DBMS Software," *Digital Technical Journal*, No. 8 (February 1989).
10. C. Mohan, K. Britton, A. Citron, and G. Samaras, "Generalized Presumed Abort: Marrying Presumed Abort and SNA's LU 6.2 Commit Protocols," *Proceedings International Workshop on Advanced Transaction Models and Architectures*, Goa, India (August–September 1996); also available as IBM Research Report RJ8684 from the IBM Almaden Research Center.
11. K. Shoens and K. Treiber, *Method for Lock Management, Page Coherency, and Asynchronous Writing of Changed Pages to Shared External Store in a Distributed Computing System*, U.S. Patent 4,965,719, IBM Corporation (October 1990).
12. The Tandem Database Group, "NonStop SQL: A Distributed, High-Performance, High-Availability Implementation of SQL," *Proceedings 2nd International Workshop on High Performance Transaction Systems*, Asilomar, CA (September 1987), pp. 60–104; also in *Lecture Notes in Computer Science* **359,** D. Gawlick, M. Haynie, and A. Reuter (Editors), Springer-Verlag, NY (1989).
13. *Teradata DBC/1012 Data Base Computer Concepts and Facilities—Release 3.1*, Document Number C02-001-05, Teradata Corporation (May 1988).
14. D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project," *IEEE Transactions on Knowledge and Data Engineering* **2,** No. 1, 44–62 (March 1990).
15. The use of the term "data sharing" in this paper is sometimes used to refer to the DB2 V4 data sharing function, which uses an SDa architecture. We do not use "data sharing" as a synonym for SDi, as in Shoens (Reference 1).
16. A "single point of failure" is a system component that, if it fails, has no backup; i.e., failure at that point causes the entire system to fail.
17. *ITSC DB2 for MVS/ESA Version 4 Data Sharing Performance Topics*, SG24-4611, IBM Corporation (1995); available from IBM branch offices.
18. These figures do not include an estimated 3 percent fixed MVS sysplex overhead.
19. The time that it takes to interact with the CF for a global lock will depend on the processor type on which the CF is running and also on the speed (and length) of the CF links.
20. C. Mohan, B. Lindsay, and R. Obermarck, "Transaction Management in the R* Distributed Data Base Management System," *ACM Transactions on Database Systems* **11,** No. 4, 378–396 (December 1986); also available as IBM Research Report RJ5037 from the IBM Almaden Research Center.
21. DB2 only uses the force-at-commit policy if there is actual "physical" inter-DB2 R/W interest on a page set or partition. The inter-DB2 interest level is dynamically tracked by DB2, as explained elsewhere in this paper.
22. A "plan" is the control structure produced during the application bind process and used to process SQL statements encountered during statement execution.

23. By "physical consistency" we mean that only one transaction at a time can be moving bits around on a given page. If multiple transactions were allowed to concurrently modify a page at the same instant in time, then the updates may interfere with one another, thus rendering the page physically inconsistent.
24. We use the term "update" to generically refer to any SQL INSERT, UPDATE, or DELETE command.
25. Repeatable read (RR) is the isolation level that provides maximum protection from other executing application programs. When an application program executes with RR protection, rows referenced by the program cannot be changed by other programs until the program reaches a commit point.
26. This technique is used to recover the page during restart, applying the "redo" log records where the LSN is greater than the LSN in the page header. It is the consequence of the DBMS not writing the updated page to disk at commit time. DB2 uses this approach.
27. C. Mohan, I. Narang, and J. Palmer, *A Case Study of Problems in Migrating to Distributed Computing: Page Recovery Using Multiple Logs in the Shared Disks Environment*, IBM Research Report RJ7343 (March 1990); available from the IBM Almaden Research Center.
28. R. Crus, "Data Recovery in IBM Database 2," *IBM Systems Journal* **23,** No. 2, 178–188 (1984).

General references

C. Carr, R. L. Huddleston, and J. Strickland, *Method and Means for the Retention of Locks Across System, Subsystem, and Communication Failures in a Multiprocessing, Multiprogramming, Shared Data Environment*, U. S. Patent 4,480,304, IBM Corporation (1985).

DB2 for MVS/ESA Version 4 Data Sharing: Planning and Administration, SC26-3269-01, IBM Corporation (1995); available through IBM branch offices.

D. Haderle and R. Jackson, "IBM Database 2 Overview," *IBM Systems Journal* **23,** No. 2, 112–125 (1984).

IBM S/390 Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex, GC28-1208, IBM Corporation (1994); available through IBM branch offices.

J. Josten, T. Masatani, C. Mohan, I. Narang, and J. Teng, *Efficient Data Base Access Using a Shared Electronic Store in a Multi-System Environment with Shared Disks*, U.S. Patent 5,408,653, IBM Corporation (April 1995).

C. Mohan, "ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes," *Proceedings 16th International Conference on Very Large Data Bases*, Brisbane, Australia (August 1990), pp. 392–405; also available as IBM Research Report RJ7008 from IBM Almaden Research Center.

C. Mohan, "A Cost-Effective Method for Providing Improved Data Availability During DBMS Restart Recovery after a Failure," *Proceedings 19th International Conference on Very Large Data Bases*, Dublin, Ireland (August 1993), pp. 368–379; also available as IBM Research Report RJ8114 from IBM Almaden Research Center.

C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Transactions on Database Systems* **17,** No. 1, 94–162 (March 1992). A different version of this paper is available as IBM Research Report RJ6649 from IBM Almaden Research Center.

C. Mohan and I. Narang, "Data Base Recovery in Shared Disks and Client-Server Architectures," *Proceedings 12th International Conference on Distributed Computing Systems*, Yokohama, Japan (June 1992).

C. Mohan and I. Narang, "Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment," *Proceedings 17th International Conference on Very Large Data Bases*, Barcelona, Spain (September 1991), pp. 193–207. A longer version is available as IBM Research Report RJ8017 from IBM Almaden Research Center.

C. Mohan and H. Pirahesh, "ARIES-RRH: Restricted Repeating of History in the ARIES Transaction Recovery Method," *Proceedings 7th International Conference on Data Engineering*, Kobe, Japan (April 1991), pp. 718–727; also available as IBM Research Report RJ7342 from the IBM Almaden Research Center.

MVS/ESA Setting up a Sysplex, GC28-1449-02, IBM Corporation (1995); available through IBM branch offices.

"Oracle Version 6.2 for Loosely-Coupled Systems," *FT Systems Newsletter*, O. Serlin, Editor, No. 101/102 (January/February 1991).

K. Shoens, *Integrated Hierarchical Locks for Data Sharing*, IBM Invention Disclosure SA8-88-0058 (1988).

Accepted for publication January 28, 1997.

Jeffrey W. Josten *IBM Software Solutions Division, Santa Teresa Laboratory, 555 Bailey Avenue, P.O. Box 49023, San Jose, California 95161-9023 (electronic mail: josten@vnet.ibm.com).*

Mr. Josten is a senior programmer with the DB2 development group at the Santa Teresa Laboratory. He was the team leader of the DB2 Version 4 data sharing development effort, and continues in that role as enhancements to the data sharing function are delivered in future DB2 releases. He joined IBM in 1985, and has been a member of the DB2 development team since 1987. His design and development activities cover a broad range of DB2 components, with emphasis on the locking and buffer management functions. He holds several software patents in the area of multisystem database sharing and is a frequent conference speaker on DB2 topics. Mr. Josten holds a B.S. degree in mathematics and computer science from the University of Wisconsin at Madison.

C. Mohan *IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: mohan@almaden.ibm.com).*

Dr. Mohan has been a research staff member at IBM's Almaden Research Center since 1981 and a member of the IBM Academy of Technology since 1992. He is currently leading the Exotica project on advanced transaction management and workflow systems. His research ideas are incorporated in numerous products (DB2, S/390 Parallel Sysplex coupling facility, SQL/DS, MQSeries, and others). Dr. Mohan received the ACM SIGMOD Innovations Award in 1996 for "innovations that have been truly outstanding and that have made a major impact in the database field." He has also received many IBM awards, including the 9th Plateau Invention Achievement Award for his patent activities. In 1997 he was honored as one of IBM's Master Inventors. He was the Americas Program Chair for the 1996 International Conference on Very Large Data Bases. He is an editor of the *VLDB Journal* and of *Distributed and Parallel Databases—An International Journal*. Dr. Mohan received a Ph.D. degree in computer science from the University of Texas at Austin in 1981 and a B. Tech. in chemical engineering from the Indian Institute of Technology, Madras in 1977.

Inderpal Narang *IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: narang @almaden.ibm.com).* Mr. Narang is a Senior Technical Staff Member in the Almaden Research Center. He joined IBM in 1981 and has been working with the database products and their multisystem coupling aspects. His key contributions have been in the architecture and algorithms of the coupling facility and DB2 data sharing in the coupled systems, and he received an IBM Corporate Award for this work. He has published papers and holds several patents in these areas. Currently, he is working on the DataLinks architecture, which extends the database management of data to files in the file systems.

James Z. Teng *IBM Software Solutions Division, Santa Teresa Laboratory, 555 Bailey Avenue, P.O. Box 49023, San Jose, California 95161-9023 (electronic mail: jteng@vnet.ibm.com).* Dr. Teng is a Senior Technical Staff Member who has worked on DB2 since 1980. He has extensive knowledge in areas of database locking, data recovery, data management, buffer pool management, and performance-related database technology. Dr. Teng is the lead architect in DB2 for the System/390 Parallel Sysplex architecture. He has obtained numerous software patents for IBM on relational database technology and has published several papers in technical journals. He is also a frequent speaker at the SHARE, GUIDE, IDUG (International DB2 User's Group), and DB2 conferences. Dr. Teng received a master's degree in computer science and a Ph.D. degree in statistics from the University of Wisconsin at Madison.

Reprint Order No. G321-5646.