Avoiding locks in DB2 Version 3.(Tutorial)

From:
Enterprise Systems Journal
Date:
December 1, 1994
Author:
Favero, Willie
More results for:
Willie Favero

Locks can ensure data integrity in DB2, but may also negatively affect performance. Until release 3 of DB2, a lock was always required every time an application needed to access any page of DB2 data. Release 3's lock avoidance technique permits a program to avoid having to use a lock under some circumstances. In version 3, when a GETPAGE request is issued, DB2 will put a latch on the page instead of a lock, to sometimes allow immediate access. Locks and latches do behave similarly, the difference is in how they are managed. A lock is maintained through the IRLM, and requires an MVS cross-memory call and a search of the hash chain. A latch, on the other hand, is managed by the Buffer Manager at the program level. Using these lock avoidance techniques may provide performance improvements, although lock avoidance should not always be used. However, it will always be attempted for applications that are bound with Cursor Stability.

Advantages And Disadvantages

Locking has fascinated users of DB2 for years. Numerous articles and presentations have described how locks can ensure data integrity but adversely affect performance. All in all, locking really has not changed much over the life of DB2. Traditionally, if an application wanted to touch a page of data in DB2, some kind of lock had to be taken, no matter how the application intended to use the page. In fact, a page could not even be examined for a qualifying row unless a lock had been obtained for that page.

This has been true in DB2 until now. DB2 Version 3 introduces a technique called lock avoidance that allows an application program to avoid using locks in certain situations. Lock avoidance, however, should not be confused with the "dirty read" DB2 users have been looking for over the last few years. A dirty read allows a SQL "reader" to look at a piece of data being changed by another application before the data has been committed. DB2 V3 still will not allow a SQL dirty read.

Although many DB2 users had expected row-level locking in Version 3, it is not there. If a locking method similar to page-level locking used in DB2 Version 2 was applied at the row level, it would never allow for the performance DB2 users have grown to expect in the 1990s. Either development of an alternate locking method or the implementation of a new locking technique that would allow DB2 to avoid locks completely was needed. The DB2 developers chose the latter. The lock avoidance technique available in DB2 V3 is actually a precursor to DB2's future row-level locking. Locking: Old And New

An application program requests data from DB2 by issuing a call to the DB2 language interface. The request is "passed" to the Database Services Address Space, which does the I/O for the application. However, before a page request can be started, DB2 must first verify that the type of

lock needed by the current page request does not conflict with any locks already held on that page. DB2 accomplishes this by checking the IMS/VS Resource Lock Manager (IRLM) to see if another application has already requested access to that page. It checks the lock requested by the second application to see if it can exist with the lock being held by the first application. Information is passed to the IRLM using MVS cross-memory services; page lock data is stored in either the IRLM address space or in Common Service Area (CSA). All are expensive resources. This data is accessed quickly by DB2 using a hash algorithm. The IRLM in DB2 V2 had only 512 hash anchor points, which degraded performance as the number of concurrent page lock requests increased. This limit however, has been increased to 16,384 in DB2 V3.

Assuming a hash entry point is available, the hash chain is followed to find the requested page. If the page is not locked or is holding a lock compatible with the type of lock being requested, access to the page is allowed. Either a row is located by DB2 and returned to the application or no row qualifies and another page request is issued. If a row does qualify, the page lock is held until after the row has been returned to the application for cursor stability or until commit for repeatable read. DB2 goes through this process before it even knows a row qualifies on the page. If, after acquiring the lock, DB2 finds there are no qualifying rows, it will simply repeat this process for the next page.

This process has changed considerably in DB2 V3. Under Version 3, when DB2 issues a GETPAGE request, it puts a "latch" on the page rather than a lock (more on latches later). This sometimes allows immediate access to the page. At the very least, it allows DB2 to examine the page for a qualifying row before a lock, if a lock is needed, is acquired for that page. Also, when a lock is taken by DB2 V3, the lock is now released before control is returned to the application.

There will still be situations in Version 3 when a lock must be used. One of DB2's criteria when considering latches is the isolation level specified at BIND time. DB2 will only consider lock avoidance if the plan or package has been bound with isolation-level Cursor Stability (CS). The semantics of Repeatable Read (RR) have not changed in Version 3. DB2 still can guarantee only that a previously read data page will return the same data if read later by holding a Shared Lock (S-Lock) on that page for the commit scope. The S-lock prevents an exclusive Lock (X-Lock) from being acquired on the page. The X-lock is required by DB2 before an application can modify the contents of the page.

Besides CS, the application must also be bound with CURRENTDATA(NO) for lock avoidance to be considered. When the plan or package is bound with CURRENTDATA(NO), DB2 knows the data needs only to be consistent (or committed) at the instant the data is read. Another application could change that piece of data immediately after the first application has read it. If the application is bound with CURRENTDATA(YES), the data must remain consistent until the cursor moves off that page.

Finally, the SQL statement must be read-only. SQL that might change the data cannot participate in lock avoidance. A SQL statement accessing the data for read-only processing should not be confused with starting the tablespace as ACCESS(RO). DB2 will not use page-level locking if a tablespace is started as ACCESS(RO) or switched to read-only access through pseudo close or with the start command. The tablespace is accessed under a single tablespace-level S-lock. This change was made to DB2 in Version 2.3 and is true despite the LOCKSIZE specified for the tablespace.

Locks Or Latches

How can DB2 V3 avoid locks? What is the difference between a lock and a latch? Page locks in Version 2 and page latches in Version 3 actually behave similarly. The major difference between

the two is how they are managed. A page lock is maintained by using the IRLM, requiring an MVS cross-memory call and a search of the hash chain in the IRLM each time a page is referenced. A page latch in Version 3 is managed by the Buffer Manager at the program level. The code path for obtaining a page latch is considerably shorter than the path to access the IRLM. Version 2.3 did use a form of latches; however, in Version 3 these are new latches not maintained by the Latch Manager and have a completely new and different purpose.

There are three basic reasons why DB2 uses a lock in Version 2. The first reason is to guarantee the page about to be read has been committed. An S-Lock cannot be obtained on a page being accessed for change with an X-Lock. DB2 will not allow a SQL statement to access an uncommitted page being modified by another user. The second reason is to prevent access to a page that is physically inconsistent. If DB2 is attempting to insert a row into a page, for example, DB2 knows there is room for the row on that page from the freespace information in the header portion of the data page. However, this freespace may not be contiguous. In that case, DB2 will compact the page at the time of the insert. While the page is being compacted, rows are being moved and map IDs are being adjusted. No one else can access this page while this is going on. Therefore, the page is locked. The final reason is repeatable read, as was previously discussed.

DB2 can attempt to read data using a latch if it knows for sure the data it is trying to access is committed and the row needs only to be consistent at the instant of the read. DB2 V3 introduces a technique that does not require a lock and use of the IRLM to determine whether the row has been committed.

Is Data Committed At Read?

If the Plan has been bound with CURRENTDATA(NO), DB2 can perform two checks to determine if the data row being read is committed, allowing the row to be accessed using a latch. They are the Commit Log Sequence Number (CLSN) and the Possibly Uncommitted (PUNC) checks. If either of these checks is successful, DB2 knows the data being read is physically consistent.

The first check DB2 performs when trying to decide if it can avoid acquiring a lock is the CLSN check. The CLSN is sometimes called the Page LSN. An LSN in Version 3 is the same as a Relative Byte Address (RBA). One byte into the header portion of each data page (see Figure 1) is the log RBA of the last time this page was changed (PGLOGRBA). There is also a control block structure used by the Buffer Manager called the Page Set Piece Control Block (PB0) that contains a list of all active Units of Recovery (UR) by pageset. A complete description of this control block can be found in the DB2 Diagnostics and Reference Manual (LY27-9603).

DB2 compares the last change RBA in the data page being accessed to the oldest active UR for that pageset in the PB0. If the last change RBA in the data page is lower than the oldest active UR, there are no active URs that can be accessing that page. If the page is not part of any active unit of recovery, it must be a committed page. Because the CLSN test is successful, DB2 does not need a lock to determine if the page is committed before reading the page. DB2 allows the application attempting the read to access the page using a page latch only.

But what if the CLSN test is unsuccessful? DB2 still has an additional test to determine if the row being read is a committed row, the PUNC test (see Figure 2). As of DB2 V3, when a row is inserted or updated, the sixth bit (counting from zero) in the first byte, PGSFLAGS, of the six-byte row prefix is set to 1. This is called the PUNC bit. DB2 will turn the bit off on a subsequent access of the row after the page has been committed. This is why it is called possibly uncommitted. The page can be committed and still have the PUNC bit set on.

If the CLSN test fails, DB2 will test the PUNC bit of the row it is about to read. If the bit is still set to off (zero), DB2 knows this row has not been changed since the last time this page was committed. Therefore, DB2 can assume this row is still committed and allow the row to be read. If the bit is set to on (1), then DB2 must acquire a lock to use this page. For a tablespace scan or an index probe, the lock is taken immediately if the PUNC test fails. The PUNC bit should not be confused with the dirty page bit (PGDIRTY), which is part of the PGFLAGS field in the data page header. PGDIRTY bit is turned on when a page is modified after an image copy. It is turned off by an image copy and has a use completely different from the PUNC bit introduced in Version 3.

Besides the PUNC bit in the row prefix, there is a new field defined in the page header, PGPUNCRA. This field is a counter of the number of PUNC bits in the page that are set to "on." If this value is zero, then the entire page is committed.

Data Consistency At Instant Of Read

What if a SQL application is accessing a data row using a latch (no lock involved), and a second SQL application wants to change that row? What is to prevent the second SQL application from taking a lock and updating the row? By design, nothing. Of course, there is more to it. The BIND Plan keyword CURRENTDATA was mentioned previously as a requirement for implementing lock avoidance. CURRENTDATA is no longer exclusively for distributed processing. It now also affects how DB2 "acts" in a local environment. CURRENTDATA tells DB2 how current, or how consistent, the data must remain while the SQL application is processing the data.

If CURRENTDATA(NO) is chosen (and this is the default if the keyword is not specified or the plan is moved from Version 2 to Version 3), DB2 assumes the data being read needs only to be consistent at the instant it is returned to the application. If another application comes in and changes that row the nanosecond after the row has been returned to the first application, DB2 does not care. However, if CURRENTDATA(YES) was chosen at BIND, then DB2 will not allow access to that row until the cursor is moved from the page. In other words, no one can access the data until the first application finishes with it. The only way to enforce this when CURRENTDATA(YES) is specified is with a lock. The application would be forced to acquire an S-lock on the page to read any of the rows from that page. Lock avoidance would not be attempted. CURRENTDATA(YES) essentially turns off the use of lock avoidance.

In some applications, this may be needed. If, for example, a cursor is defined as read-only because it contains an ORDER BY clause, an UPDATE or DELETE WHERE CURRENT OF clause would not be allowed. Instead, an UPDATE or DELETE SQL statement independent of the cursor would need to be used. This is a fairly common practice. In Version 2, the cursor would be fetching rows under an S-Lock. When it needs to perform the update, the S-Lock is promoted to an X-Lock. Because this all happens within the same application, that application would be first to acquire the X-Lock. However, in Version 3, this would not be true.

What Is Lock Avoidance Doing?

Currently, the latches maintained by the Buffer Manager are "wait forever" and DB2 practices a form of deadlatch (similar to deadlock) prevention. This is accomplished by requesting latches in specific order and by managing the latches at the program level. Latch timeouts and deadlatches should not occur. However, an application could have to wait on a latch. Latch wait times are already reported in Version 2 performance statistics. These latch times should not be confused with the latches used for lock avoidance. Lock avoidance uses the Buffer Manager, not the latch manager, and produces new wait times statistics.

Version 3 modifies some existing Instrumentation Facility Interface records (IFC1Ds) and introduces a new series of IFCIDs reporting wait time events of page latches. A new IFCID pair reports suspend and resume page latch events. IFCIDs 226 (page latch suspend) and 227 (page latch resume) have been added to Performance Class 4 (reads to and writes from the buffer and Environmental Descriptor Manager (EDM) pools), Accounting Class 3 (elapsed wait times in DB2) and Class 8 (elapsed wait times in DB2 for packages and DataBase Request Modules (DBRMs)), and Monitor Class 3 (wait times for I/O and locks) traces.

There are also new fields in the accounting record that give accumulated page latch wait time and the number of page latch waits. Field QWACAWTP in the accounting record reports the accumulated page latch wait times. There is an associated counter field for waits. QWACAWTP is derived based on 226 and 227 being in accounting and Monitor Class 3. This information is based completely on the specifications. Page latch wait time is separate from previously reported latch wait times.

New' IFCIDs contain the lock statistics and what has been updated in existing IFCIDs with latch information. IFCID 0218 (Commit_LSN test) is written or Performance Class 6 trace is on. IFCID 0223 (successful lock avoidance) is written when Performance Class 7 trace is active.

There are four additional IFCIDs (51, 52, 56 and 57) included in Accounting Classes 3 and 8, and Monitor Class 3 traces that report latch wait times. These IFCIDs have nothing to do with lock avoidance.

Conclusion

Think of the potential performance improvements if long-running queries (say in QMF, where a FOR FETCH ONLY is generated for every select that is run) or a batch application performing a tablespace scan suddenly had a code path thousands of instructions shorter. That is what will happen every time DB2 V3 can avoid a lock. Some lock waits could also be avoided. However, do not rush in blindly. Remember, there is a slight exposure for applications with READ ONLY cursors that issue an independent SQL UPDATE or DELETE. Also, keep in mind that the default for CURRENTDATA is NO, so lock avoidance will always be attempted for applications bound with Cursor Stability. And finally, remember this is just the beginning of lock importance in DB2!

ABOUT THE AUTHOR

Willie Favero is a software consultant with BMC Software, Inc. Before coming to BMC, he spent 10 years with IBM Corp. where he taught courses on DB2 V3 transition, recovery, administration, design and performance. At IBM he also contributed to "red books" and authored educational course materials. Prior to joining IBM, he held various positions as an application programmer/analyst and systems programmer in IMS, MVS, VTAM and NCP. BMC Software Inc., 2101 Citywest Blvd., Houston, IX 77042. (713) 918-8000.