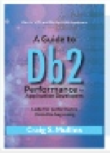

ENTERPRISE
Tech
JOURNAL
Formerly z/Journal
THE MAGAZINE FOR IT STAFF IN THE WORLD'S LARGEST HYBRID DATA CENTERS

October / November 2012

zData Perspectives

What Can You Do to Avoid DB2 Locking Problems?

by Craig S. Mullins

DB2 lock contention issues can be frustrating problems to investigate and debug. Before blaming DB2 (as is the usual response), try to answer the following questions to help identify the cause of the locking problems:

- Has the application ever run without problems?
- Have the lock timeouts or deadlocks started recently?
- What level of DB2 are you running?
- Does the problem only occur at certain times?
- What has changed on the system (e.g., number of users, number of applications, amount of data, Program Temporary Fixes [PTFs] to DB2 or any other relevant software, etc.)?
- What has changed in the application (e.g.,

pre-determined period of time, it will be terminated. When a process is terminated because it exceeds this period of time, it's said to timeout. In other words, a timeout is caused by the unavailability of a given resource.

To minimize lock timeouts, design your application programs with locking in mind from the start. Limit the number of rows accessed by coding predicates to filter unwanted rows. Doing so reduces the number of locks on pages containing rows that are accessed but not required. Also, design update programs so the update is issued as close to the COMMIT point as possible. Doing so reduces the time that locks are held during a unit of work, which also reduces timeouts (and deadlocks).

Speaking of COMMITs, it's important to design all your batch programs with a COMMIT strategy. A COMMIT externalizes the modifications that occurred in the program since the beginning of the program or the last COMMIT. A COMMIT makes sure all modifications have been physically applied to the database, thereby ensuring data integrity and recoverability. Failing to isolation level, concurrent executions, volume of data, etc.)?

When one application program tries to read data that's in the process of being changed by another, the DBMS must control access until the modification is complete to ensure data integrity. Most DBMS products—DB2 included—use a locking mechanism for all data items being changed. Therefore, when one task is updating data on a page, another task can't access data (read or update) on that same page until the data modification is complete and committed. When multiple users can access and update the same data at the same time, a locking mechanism capable of differentiating between stable data and uncertain data is required. Stable data has been successfully committed and isn't involved in an update in a current unit of work. Uncertain data is currently involved in an operation that could modify its contents.

This is a simplified discussion of locking and isn't intended to explain all the nuances of locking in DB2 for z/OS. Instead, let's look at techniques for remediating locking problems.

Lock timeouts are perhaps the most vexing issue encountered by DB2 professionals. The longer a lock is held, the greater the potential impact to other applications. When an application requests a lock that's already held by another process, and the lock can't be shared, that application is suspended. A suspended process temporarily stops running until the lock can be acquired. When an application has been suspended for a

code COMMITs in a data modification program is what I like to call "Bachelor Programming Syndrome"— in other words, fear of committing. This can cause lock timeouts and lock escalation.

There are techniques available to DBAs to minimize lock timeouts. When an object is being accessed concurrently by multiple processes, consider using the MAXROWS option of the CREATE TABLESPACE statement to cause fewer rows to be stored on a single page. The fewer rows per page, the less intrusive page locking will be because fewer rows will be impacted by a page lock.

Deadlocks also can cause problems. A deadlock occurs when two separate processes compete for resources held by one another. For example, a deadlock transpires when PGMA has a lock on PAGE1 and wants to lock PAGE2 but PGMB (at the same time) has a lock on PAGE2 and wants a lock on PAGE1. One of the programs must be terminated to allow processing to continue. One technique to minimize deadlocks is to code your programs so that tables are accessed in the same order. By designing all application programs to access tables in the same order, you reduce the likelihood of deadlocks.

Locking is a complex issue and can be at the root of many performance problems. But if you follow the guidance offered here, you can reduce the frequency of locking issues in your shop.

From Enterprise Tech Journal, October / November 2012.

DB2PORTAL.com