Rule DB2-311: Deadlocks were detected

Finding: Deadlocks were detected in the DB2 interval statistics data.

Impact: This finding can have a MEDIUM IMPACT, or HIGH IMPACT on the performance of the DB2 subsystem.

Discussion: A deadlock is an unresolvable contention for the use of a resource such as a table or an index. A deadlock occurs when two or more application processes hold locks on resources that the others need and without which they cannot proceed.

The *DB2 Administration Guides*[1] give the following example of a deadlock situation:

- Jobs EMPLJCHG and PROJNCHG are two transactions. Job EMPLJCHG accesses table M, and acquires an exclusive lock for page B, which contains record 000300.
- Job PROJNCHG accesses table N, and acquires an exclusive lock for page A, which contains record 000010.
- Job EMPLJCHG requests a lock for page A of table N while still holding the lock on page B of table M. The job is suspended, because job PROJNCHG is holding an exclusive lock on page A.
- Job PROJNCHG requests a lock for page B of table M while still holding the lock on page A of table N. The job is suspended, because job EMPLJCHG is holding an exclusive lock on page B. The situation is a deadlock.

After a preset time interval (the value of DEADLOCK TIME), DB2 will detect the deadlock situation. DB2 can roll back the current unit of work for one of the processes, or DB2 can request a process to terminate.

It is possible for two processes to be running on separate DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

[1] With DB2 Version 9, this example is contained in the DB2 UDB for z/OS Version 9: Performance Monitoring and Tuning Guide.

Either of these actions frees the locks and allows the remaining processes to continue. However, the rollback of the current unit or work, or termination of a process can seriously affect performance. Not only does the performance of the deadlock processes suffer, but significant system overhead can be incurred.

CPExpert compares the QTXADEA variable in DB2STATS (the number of deadlocks encountered) with **the QTXADEA** guidance variable in USOURCE(DB2GUIDE). CPExpert produces Rule DB2-311 when the number of deadlocks encountered exceeds the value specified by **the QTXADEA** guidance variable.

The default value for **the QTXADEA** guidance variable is 0, indicating that CPExpert should produce Rule DB2-311 when any deadlocks were detected.

The following example illustrates the output from Rule DB2-311:

```
RULE DB2-311: DEADLOCKS WERE DETECTED

   A deadlock occurs when two or more application processes each hold
   locks on resources that the others need and without which they cannot
   proceed.  DB2 scans for deadlocked processes at regular intervals,
   with the scan interval set by the DEADLOCK TIME on Installation Panel
   DSNTIPJ.  Upon detecting a deadlock, DB2 can roll back the current unit
   of work for one of the processes or request a process to terminate.
   Deadlocks can significantly affect performance and should be avoided.
   This situation occurred during the intervals shown below:

                                  LOCK       LOCK       LOCK
   MEASUREMENT INTERVAL         REQUESTS  SUSPENSION  TIMEOUTS  DEADLOCKS
   21:44-21:59, 08SEP1998       2,819,800       43         19         5
```

Suggestion: If application processes encounter deadlocks, CPExpert suggests that you consider the following alternatives:

- If statistics trace class 3 is active, DB2 writes a “Unit of Work involved in deadlock” trace record (IFCID 0172). Deadlocks do not occur often, so little overhead is involved in producing this trace record. This trace record should be analyzed to identify the unit of work involved in the deadlocks. This information is located in MXG file T102S172.

- As mentioned earlier, DB2 scans for deadlocked processes at regular intervals, as specified by the DEADLOCK TIME on Installation Panel DSNTIPJ. The default for the DEADLOCK TIME is five seconds.

IBM recommends that this default be adjusted so DB2 can detect deadlocks as quickly as possible. IBM recommends that, in most cases, the default be changed to specify a value of 1.

- If a task is waiting or is swapped out and the unit of work has not been committed, then it still holds locks. When a system is heavily loaded, contention for processing, I/O, and storage can cause waiting. Consider reducing the number of initiators, increasing the priority for the DB2 tasks, and providing more processing, I/O, or storage resources.

- Make sure that the IRLM has a high MVS-dispatching priority. IBM recommends that IRLM come next after VTAM and before DB2. If you are running in Goal Mode, you should assign IRLM to SYSSTC, so it will have high dispatching priority.

- Access data in a consistent order: When different applications access the same data, try to make them do so in the same sequence. For example, make each application access rows 1,2,3,5 in that order. In that case, the first application to access the data delays the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.

- Commit work as soon as is practical. To avoid unnecessary lock contentions, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications. To prevent unsuccessful SQL statements (such as PREPARE) from holding locks, issue a ROLLBACK statement after a failure. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature.

- Retry an application after deadlock or timeout. Include logic in a batch program so that it retries an operation after a deadlock or timeout. That could help you recover from the situation without assistance from operations personnel. Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or timeout occurred.

- Close cursors. If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point. Use the CLOSE CURSOR statement as soon as possible in your program, to release those locks and free the resources they hold.

- Bind plans with ACQUIRE(USE): That choice is best for concurrency. Packages are always bound with ACQUIRE(USE), by default. ACQUIRE(ALLOCATE) gives better protection against deadlocks for a high-priority job; if you need that option, you might want to bind all DBRMs directly to the plan.

- Bind with ISOLATION(CS) and CURRENTDATA(NO) Typically: ISOLATION(CS) lets DB2 release acquired locks as soon as possible. CURRENTDATA(NO) lets DB2 avoid acquiring locks as often as possible. After that, in order of decreasing preference for concurrency, use these bind options:

- ISOLATION(CS) with CURRENTDATA(YES), when data you have accessed must not be changed before your next FETCH operation.

- ISOLATION(RS), when rows you have accessed must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.

- ISOLATION(RR), when rows you have accessed must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.

- Use ISOLATION(UR) Cautiously: UR isolation acquires almost no locks. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

- If you can define more ECSA, then start the IRLM with PC=NO rather than PC=YES. You can make this change without changing your application process. This change can also reduce processing time.

- Alternatively, you can revise the application so that it issues COMMIT more frequently.

You should review the sections titled "Tuning Your Use of Locks" in IBM's DB2 *Administration Guides*[2] for more suggestions from IBM on how to minimize or eliminate the negative effects of locks.

- You can alter CPExpert's analysis by modifying the **QTXADEA** guidance variable in USOURCE(DB2GUIDE).

Report: CPExpert will produce a report relating to deadlocks. This report is produced if DB2 accounting data is available, and if %LET DB2ACCTX=Y; is specified in USOURCE(DB2GUIDE) to tell CPExpert that the DB2 accounting data is available.

This Deadlock Report will provide information about the holder of the resource holder and the resource wanter. Information will show the

[2]Note that this section is titled "Lock tuning" in DB2 UDB for z/OS Version 8

Correlation ID, the Plan Name, and the Connection ID for both the resource holder and the resource wanter.

Reference: DB2 for OS/390 Version 3: Administration Guide
Section 7.8.1 (Aspects of Transaction Locking)
Section 7.8.4 (Tuning Your Use of Locks)

DB2 for OS/390 Version 4: Administration Guide
Section 5.7.3 (Basic Recommendations to Promote Concurrency)
Section 5.7.4 (Aspects of Transaction Locks)
Section 5.7.5 (Tuning Your Use of Locks)

DB2 for OS/390 Version 5: Administration Guide
Section 5.7.3 (Basic Recommendations to Promote Concurrency)
Section 5.7.4 (Aspects of Transaction Locks)
Section 5.7.5 (Tuning Your Use of Locks)

DB2 for OS/390 Version 6: Administration Guide
Section 5.7.3 (Basic Recommendations to Promote Concurrency)
Section 5.7.4 (Aspects of Transaction Locks)
Section 5.7.5 (Tuning Your Use of Locks)

DB2 UDB for OS/390 and z/OS, Version 7: Administration Guide
Section 5.7.3 (Basic Recommendations to Promote Concurrency)
Section 5.7.4 (Aspects of Transaction Locks)
Section 5.7.5 (Tuning Your Use of Locks)

DB2 UDB for z/OS Version 8: Administration Guide
Chapter 30. Improving concurrency
Aspects of Transaction Locks
Lock tuning

DB2 UDB for z/OS Version 9: Performance Monitoring and Tuning Guide
Chapter 8. Improving concurrency
Aspects of Transaction Locks
Options for tuning locks

DB2 10 for z/OS: Managing Performance
Chapter 27. Programming for concurrency
Options for tuning locks

DB2 11 for z/OS: Managing Performance |
Chapter 17. Concurrency and locks |