CA IDMS Deadlock Analysis

OVERVIEW

Whenever concurrent updating of a database is to occur it is necessary to have in place a locking mechanism that prevents 2 or more transactions from concurrently updating the same record occurrences. When contentions for a lock occur, typically the first transaction requesting the resource gets access to it while subsequent transactions wait for the first transaction to release the lock. This can result in a slowing of processing and in extreme situations some of the waiting tasks may abend due to exceeding stall intervals or due to the transactions involved getting into a deadly embrace, more commonly referred to as a deadlock.

Figure 1 provides an example of a simple wait scenario where transaction 1 modifies a database record occurrence followed by an attempt by transaction 2 to access the same record occurrence. In this situation transaction 2 will wait until transaction 1 releases its lock of the record by issuing a FINISH or COMMIT command.

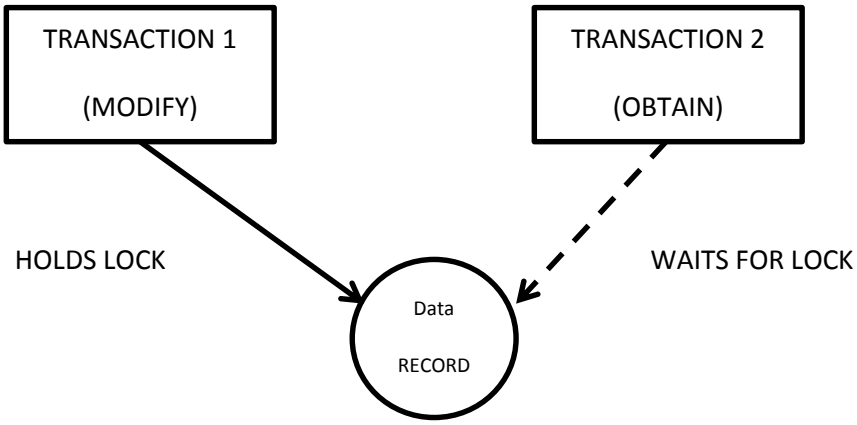

FIGURE 1

Based on the sequence in which transactions access database records it is possible that two or more transactions each hold resources required by the other transactions. When this occurs a deadly embrace or deadlock occurs. This requires that one of the transactions be rolled out so that the database resources it controls can be released allowing the other transaction to continue processing.

In Figure 2 we see that transaction 1 holds a lock on record A and needs to set a lock on record B. However transaction 2 holds the lock on record B so transaction 1 must wait. Transaction 2 then tries to lock record A but must wait because transaction 1 already holds a lock on that record. The result is a deadlock condition which can only be resolved by ending one of the transactions.

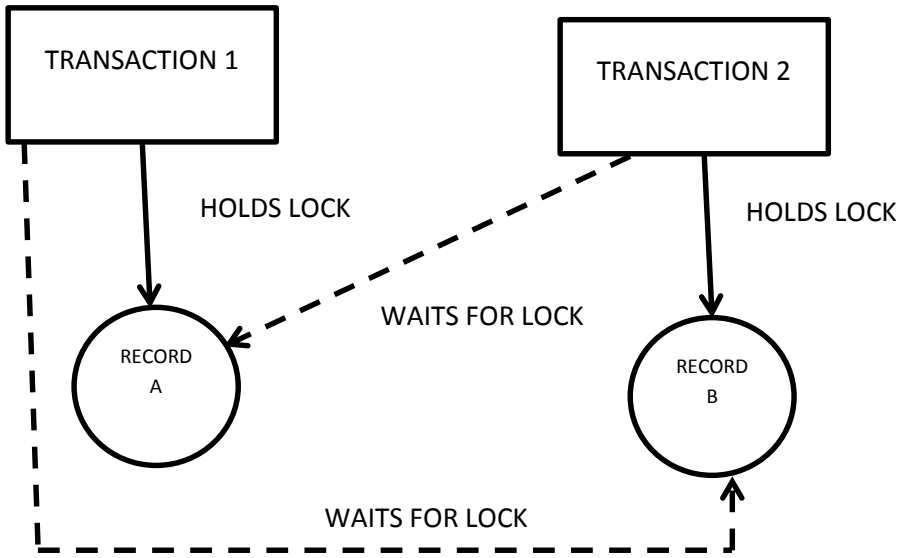

Figure 2

LOCKING ERROR MESSAGES

When a deadlock condition occurs, a cluster of messages will be written to the CV's log with information defining the tasks and transactions involved with the locking contention and the resources on which the contention occurred. You will receive a DC001000 message for each task involved in the locking contention. The DC001000 messages for all of the tasks involved is then followed by a DC001002 message indicating which task's transaction was rolled out so that the deadlock condition could be resolved.

DC001001 messages can also be optionally generated to provide additional data that can be used to analyze deadlock situations. At least two DC001001 messages will be generated immediately following the related DC001000 messages. The first of these messages is a header line followed by the actual data lines. The number of data lines generated will depend on the number of run-units (transactions) the task is associated with. To generate the DC001001 messages it is necessary for the CV to specify DEADLOCK_DETAILS=ON in its SYSIDMS file at start-up.

Figure 3 shows an example of a deadlock message cluster for a simple deadlock occurrence.

```
DC001000 T:852690 APPC  P:APXPO031 C:DEAD WAITING ON R:LTXNLOCK 00042009 00397442
DC001001  Txn/RU ID   RU  NAME    SUBSC     User ID    FE - ID1  FE - ID2  FE - ID3  FE Tskcd
DC001001  451537594  APXPO031  APXSS100  ****
DC001000  T:852711 ADS2 P:APXD1476 C:DEAD WAITING ON R:LTXNLOCK 00052008 008BDB94
DC001001  Txn/RU ID   RU  NAME    SUBSC     User ID    FE - ID1  FE - ID2  FE - ID3  FE Tskcd
DC001001  451537611  APXD1476  APXSS100  C087415
DC001002  T:852711 ADS2 P:APXD1476 C:DEAD DEADLOCKED ON R:LTXNLOCK 00052008 008BDB94
```

Figure 3

Each DC001000 message represents a task that is a participant in the deadlock and is waiting for a resource. The first 4 fields on the message provide the task id of the waiting task, the task code that invoked the task, the program name that CA IDMS that is being executed, and the status of the task. The last field following the literal ‘R:’ provides the resource upon which the task is waiting. The provided value is broken down as follows.

R: <Type><wwwwxxyy zzzzzzzz>

Type -	LTXNLOCK	Local transaction lock
	GTXNLOCK	Global transaction lock
	PAGELOCK	PAGELOCK

<wwwwxxyy zzzzzzzz>

<wwww>	Page group
<xx>	Code identifying the value in **<**zzzzzzzz**>**

'00'	<zzzzzzzz> contains a dbkey (Record lock)
'20'	<zzzzzzzz> contains a page number (Space Management lock)
'80'	<zzzzzzzz> contains an area's low page (Area lock)
'C0'	<zzzzzzzz> contains an area's low page (Transient lock)
'01'	<zzzzzzzz> contains a page number (Page lock)

<yy>	Dbkey radix

The DC001001 message provides further information to assist in the description of the task that is waiting on the resource described in the preceding DC001000 message. The message will contain the transaction id of the run-unit involved, the program name if present that was within SUBSCHEMA-CONTROL when the un-unit was bound, and the subschema that the run-unit is referencing. One DC001001 message will be displayed for each run-unit associated with the waiting task.

Finally, a DC001002 message will be generated to identify the task that was aborted to resolve the deadlock. The contents of this message are the same as those provided on a DC001000 message but the specified resource can be interpreted as the resource that was the final item that completed the deadlock scenario.

DEADLOCK ANALYSIS

It is extremely rare that the resolution of deadlocks can occur by changing any of the system-level definitions of a CV. However if a CV is experiencing excessive deadlocks the CV's sysgen should be examined for the following statements.

DEADLOCK DETECTION INTERVAL IS 1
TICKER INTERVAL IS 1

Specifying a value of '1' for each of the options will ensure that CA IDMS will search for and resolve deadlock conditions as soon as possible. The longer a deadlock condition exists the

greater the possibility that other tasks will become involved with the deadlock, further aggravating the situation.

DEADLOCK conditions are almost always the result of the design of the database structure and timing issues associated with the way the applications navigate that structure. As transaction volume increases so does the likelihood that the points at which lock contentions can occur. This can result in deadlock occurrences becoming more common, especially during periods of peak processing.

In an environment that permits concurrent updating of a database it is to be expected that some deadlock conditions will occur. It is when this number becomes excessive that analysis of the deadlocks is warranted, followed by some type of database or application modification designed to reduce or minimize a point of contention.

The best way to start a deadlock analysis is to scan the messages from all of the deadlock occurrences looking for common or repeating values. First, look at the resources reported within the DC001000 and DC001002 messages for each deadlock occurrence. If a single resource is present in a large percentage of the deadlocks you have probably located a main point of contention within the environment.

For example, if you find many DC001000 messages that report the same dbkey it's very possible that the database contains a record type for which only a single occurrence exists within the database and a large number of application transactions are required to update that record occurrence. The DC001000 message does not tell you the type of record represented by the dbkey so the next step is to run the PRINT PAGE utility to list the page number represented by the dbkey. Assuming a standard radix point of x'08' on the DC001000 message, the page number would be the first 3 bytes of the specified dbkey. From the PRINT PAGE listing you can determine the type of record represented by the dbkey and the contents of that record occurrence such as key values.

Once you have identified the resource you can use the program name and the run-unit name from the DC001000 and DC001001 messages to point you to the programs that use that record type and have some point of lock contention within their logic.

Figure 4 shows a very innocent looking piece of code that can result in serious deadlock conditions within an application.

```
OBTAIN CALC ORDER-CTRL.
PERFORM IDMS-STATUS.
MOVE NEXT-ORD-NUM TO ORDER-NUMBER.
ADD +1 TO NEXT-ORD-NUM.
MODIFY ORDER-CTRL.
PERFORM IDMS-STATUS.
```

Figure 4

Every time that an order is added to the database it is necessary that a program read the single ORDER-CTRL record occurrence to get and assign the next order number to the new order. The program must then increment the NEXT-ORD-NUM field and update the ORDER-CTRL record. This type of record is often referred to as a One-Of-A-Kind or OOAK record.

As volume increases the more frequently this record occurrence gets updated and lock contentions can occur. To resolve this situation the application might be modified to single-thread the updating of the ORDER-CTRL record through some means external to record locking such as by bracketing the above code example with ENQUEUE/DEQUEUE commands. Another option would be to modify the application to use some alternative method of assigning new order numbers.

In some scenarios the repeating resource within the deadlock members may not represent a dbkey but a page number. If the value of the <xx> portion of the DC001000 message is x'20' there is a contention attempting to manipulate the amount of free space on the specified database page. Use the provided page group and page number to determine the area in which the page resides. The PRINT PAGE utility can again be used to list the contents of the page and to identify the types of records on that page. This information will give you an idea of what type of database activity you should look for within the programs specified on the DC001000 and DC001001 messages.

Common causes of this type of lock contention are applications where large volumes of records are written to a single CA IDMS queue using PUT QUEUE commands or where records defined with a location mode of DIRECT are stored to a database area using a suggested database key of -1. In these situations it not uncommon to see the value of the repeating resource in the DC001000 message to slowly increase through the processing window. Single-threading the function or finding an alternative database structure would be required to resolve these conditions.

If the value of the <xx> component of the DC001000 message is an x'80' you have contention for a usage mode against an area. Use the provided page group and page number to identify the areas involved. Then inspect the indicated programs to determine the usage modes specified by each program to identify the point of contention.

In some scenarios the repeating value within the DC001000 and DC001001 messages may not be a resource but a program or run-unit name. If you determine that one or more programs are always involved in a deadlock occurrence there is a good probability that there is some logic within those programs that results in a contention point. Start by again using the PRINT PAGE utility to list the pages for the page numbers or dbkeys provided in the resource portion of the DC001000 messages. Use these listings to identify the types of records represented by the provided dbkeys or to determine the types of records involved with contention for a space management lock.

When the deadlocks involve the same programs but varying resources it may be necessary to chart out the sequence of DML commands within the programs involved and look for points where the record types involved intersect. These types of scenarios typically require the involvement of application staff that is familiar with the applications to accurately identify the sequence of DML commands leading to the point of contention.

Another resource that might be used to identify the sequence of the DML commands used prior to the point of contention is JREPORT08. By customizing the JREPORT08 execution to only list the transactions ids from the associated DC001001 messages you can determine the sequence of DML commands and records involved leading to the deadlock condition. You should be aware that these JREPORT08 listings may be extensive based on the number of transactions involved with the deadlock and the amount of work performed by each transaction.

Resolution of these last types of deadlock conditions depends on the offending logic that is discovered and the requirements of the application. In most cases modification of the logic of one or more of the programs involved will be necessary to minimize the number of deadlock occurrences.

GENERAL RECOMMENDATIONS

Although each deadlock scenario represents a unique set of conditions there are a number of considerations that should be observed to minimize their impact on a production environment.

- Avoid running batch updates during online windows.

  Batch updating tends to affect greater portions of a database than do online tasks. This implies that more records will be locked during the processing. If it is necessary to run batch updates during the online window use COMMIT logic within the batch logic to reduce the number of database records locked.

- Minimize the amount of work performed by online transactions.

  Whenever possible, an online transaction should minimize the amount of work performed. The greater the amount of work performed per functional key-stroke the longer it will take to complete the transactions and the more record locks that will be generated. Each of these conditions increases the potential of locking conflicts with other concurrent transactions.

- Avoid creating database hotspots within an application.

  Examine the database and application design to identify one spot within the database that numerous transactions will have to update concurrently. As volume increases these locations within the database become a more likely bottleneck which will eventually lead to deadlock conditions. Some examples of these hotspots include but are not limited to OOAK records, storing records with a DIRECT location mode and a suggested key of -1, generating DC queue records to a single queue, or constantly updating a single point within an index set.

- Avoid retaining resources across transactions or interactions with a terminal operator.

  The longer a database lock is held, the greater the possibility that it will be required by a concurrent transaction. Unless unavoidable, record locks should only be retained for the life of a transaction and should never be held across interactions with a terminal operator.

  Longterm locks are record locks that can be set such that they are retained at the conclusion of a transaction and must be released during some later transaction on the associated terminal. These types of locks should be used infrequently and only when absolutely necessary to the application.

  Conversational tasks should also be avoided whenever possible. A conversational task occurs when a program issues a MAPOUTIN command. The MAPOUTIN command sends

data back to the terminal operator and the task waits for the operator's response. If a database transaction is open at the point of the MAPOUTIN command it remains active during the wait for the operator. All outstanding record locks are also retained which can result in conflicts with tasks subsequently issued from other terminals.

- Stress test all changes before production implementation.

  Any modification to an update process that changes the sequence or volume of database records that are accessed should undergo some level of stress testing against a volume of transactions similar to the greatest anticipated load. This may be through the use of a third-party utility like TPNS or some in-house procedure to determine if any points of lock contention have been created.

  Since the occurrence of deadlocks is usually a timing situation any change to the CA IDMS CV environment should be a reason to run a stress test. Although very rare, the changing of sysgen options, DMCL characteristics, or migration to a new release of CA IDMS have in the past been seen to result in new deadlock occurrences because of subtle changes in the timings related to an application.

- Insure deadlocks are resolved as quickly as possible.

  When concurrent updating of a database is allowed it is inevitable that some deadlocks will occur. The system definitions of all CVs should specify DEADLOCK DETENTION INTERVAL IS 1 and TICKER INTERVAL IS 1 so that CA IDMS will be able to resolve deadlock conditions as quickly as possible.

Deadlock analysis is truly more of an art than a science which requires the participation of a site's application staff and not just the Database Administrator. Although the DBA may do the initial analysis of the messages generated by CA IDMS it will frequently require the application developer's specialized knowledge of the application code to interpret, identify, and rectify any underlying problems.