Monitoring CA-Ingres Deadlocking

by Dennis Adams, Senior Technical Consultant, Common Sense Computing (UK) Ltd [1]

This article discusses the issues and problems associated with monitoring Deadlocks in a CA-Ingres DBMS environment.

1. What are Deadlocks ?

Deadlocks may occur when a user (User A) wishes to gain control of a resource which is locked by a second user (User B). This is a **blocking lock**, and simply results in User A being put into a lock wait state, until the resource is free.

However, if User B is also waiting on resources, which have been granted to User A, we have a deadlock situation. Neither user is able to proceed until the other is moved.

The above is a simple **"2-way" deadlock**, although it is possible for 3 or more users to have a multi-way deadlock situation.

In either case, it is necessary for one user to be "rolled back", so that the resources are freed. Deadlock detection is done on a "round-robin" basis. The implication of this is that the user being rolled back will not necessarily be the last one to be blocked.

Monitoring Implications

The objectives of monitoring deadlocks are generally as follows:

- Identifying the session(s) and therefore the users who are being affected by deadlocks
- Identifying the session which is "rolled back", and the other session which was involved in the locking.
- Identifying (if applicable) the sections of program which were running when the deadlock occurred, for both sessions. In practice, the only way to identify the section of program which was being executed would be to identify the actual SQL statement which was being run, and cross-reference this to the source code.
- Determine whether the deadlock was caused by inappropriate code design, by CA-Ingres tuning parameters, or by a combination of both.

2. What Causes Deadlocks?

User Deadlocking

Deadlocks could be caused by users running modules of code which attempt to access 2 tables in different sequences. For example, one program frame could read (and lock) a customer table page, and

[1] **Dennis wrote this article in 1998 when he was UK Technical Consultant for Common Sense Computing (UK) Limited. It originally appeared on the Common Sense Computing Website – http://www.comsense.com. This has now been absorbed by the US Parent Company – Quest Software, http://www.quest.com.**

then attempt to update a transaction table page. A second program (run by another user) could attempt to read a transaction table page and then attempt to update a customer table page.

The above circumstances cause deadlocking only in the context of a Multi-Statement-Transaction or ("MST"), since locks are held on a resource until the statement accessing them is committed or rolled back. In an event-driven Windows4GL (CA-OpenROAD) environment, it is more common for users to enter a frame with uncommitted statements open (and therefore locks held). Therefore, there is a greater frequency of MSTs, and a consequent danger of deadlocking. Note that by default CA-Ingres locking is at PAGE level, so the locks may not be on the same data rows, they just need to reside on the same page, and therefore the same page is required at the same time by both users.

Resource Limitations

The previous example is what might be termed a "user deadlock" - 2 users simultaneously trying to access the same pages of data. However, deadlocking could also occur even if the same page is not being accessed, due to lock escalation. This happens when the Ingres system itself runs out of locking resources, and compensates by changing the type of lock.

For example, there is a maximum limit to the number of locks which a session may hold. Each page being locked requires a single entry in the lock list for that user session. Supposing a session reaches this maximum, but wishes to take out more locks (as in a complex MST, for example). It may already have locks on 10 pages of table "X", on 5 pages of table "Y", and on 12 pages of table "Z", but be unable to take out any further locks.

CA-Ingres attempts to get round this problem by replacing existing locks with "cheaper" ones. For example, if the 12 locks on pages of table "Z" were replaced by a single table-level lock on the entire table (which requires only one lock entry), this would have the effect of locking the required resource (and more besides), at the same time releasing 11 lock entries which could be used elsewhere. This is lock escalation.

When lock escalation occurs, it is possible for the table-level lock to be blocked, and deadlock could occur. Having table-level control may be an efficient use of resources, but cause havoc in operational terms.

Lock escalation of this type could occur for any one of several reasons: insufficient locks per table for any session (defined by the "maxlocks" parameter, often in ING_SET), insufficient locks per session (as defined in the CA-Ingres locking configuration file) , or insufficient locks in the locking system as a whole (defined in the Ingres locking configuration file). Any of these could lead to lock escalation.

Monitoring Implications

Since deadlocks could be caused by user code and/or resource limitations causing escalation, the following data would need to be monitored in order to identify the causes of deadlocking:

- Blocking locks, to identify potential deadlocks.
- Use of CA-Ingres resources, to identify potential escalation.
- The frequency of deadlocks occurring to cross-reference with the above.

The following sections examine how deadlocks are identified and processed by the Ingres system, and what information is made available when they occur.

3. How are Deadlocks Identified?

Deadlock detection starts whenever a lock BLOCK occurs. So every blocking lock will result in Ingres running its deadlock detection algorithm, to determine if the block is caused by a deadlock situation. Note that deadlock detection checking needs to scan all lock lists. In a VAX/VMS cluster environment, this means that all members of the cluster need to be checked whenever a blocking lock occurs. This is an important performance issue in VAXclusters.

One of the recommended ways to look for deadlocks with I/Watch is to identify blocking locks, and extrapolate this information, since all deadlocks start life as blocking locks. A blocking lock could occur for a (relatively) long period of time, compared with the deadlock processing itself.

4. How are Deadlocks Handled?

When a deadlock situation is detected by CA-Ingres, one of the CA-Ingres sessions is marked with a DEADLOCK state flag and the query is rolled back.

Note that a deadlock is between two or more users sessions, but only one of them is rolled back. Therefore, the other session may be unaware of (and unaffected by) the deadlock. The session being rolled back is kept with the DEADLOCK state flag for as long as the process takes to rollback, which may be very brief.

Therefore, although the deadlock state could conceivably be detected, it will only persist for a short period of time.

5. How are Deadlocks Recorded?

Apart from the session deadlock state, the fact that a deadlock has occurred is recorded in the server statistics. These figures are reset to zero when the server is re-started.

If the deadlock is due to lock escalation, the deadlock is recorded as such in the Ingres error log (errlog.log). However, this is done after the deadlock has been detected, and in many monitoring situations this is too late to take further action.

Deadlocks due to lock state changes (e.g. changing a lock from "share" to "write") are also always recorded in the error log.

However, deadlocks which are "user" deadlocks are not, as a general rule, recorded in the error log file.

Monitoring Implications

The previous sections suggest the following strategy for monitoring deadlocks:

- Monitor blocking locks to identify locking which is occurring, and may result in deadlocks.
- Monitor the deadlock counter in the server process.
- Monitor occurrences of deadlocks in the CA-Ingres error log file to determine resource-driven deadlocks.

Note that very frequent interrogation of CA-Ingres for locking information can have the effect of imposing an unacceptable load on the host machine.

Attempts to monitor the DEADLOCK state in a server thread are impractical, since it only exists for a very short period of time, while the rollback is occurring.

6. User Program Interaction

When a user program is notified of a deadlock situation, (ie. the program's own query is being rolled back) it will often notify the user and then attempt to re-try the query. The exact action being performed can be coded by the application developers.

Monitoring Implications

If it is known that the application software re-tries deadlocked transactions, it would be possible to monitor in the following way:

- Identify errors from the error log, and extract the server and session information, after the deadlock has occurred.
- Immediately cross-check the server session information to find the session.
- Extract the query which the session is currently executing, which may well be part of a re-try of the previous deadlocked transaction.

7. Monitoring Deadlocks with I/Watch

I/Watch is Common Sense Computing's CA-Ingres, Oracle 7 and Sybase performance monitor. It monitors a host of variables related to database or operating system performance, including some with relevance to deadlocking. I/Watch can be used to draw graphs to monitor in real time the overall level of deadlocking, and visually compare it with usage of CA-Ingres locking resources.

With its powerful scripting language, I/Watch can also be used to detect defined exception conditions (such as a deadlock) and initiate appropriate further action (such as recording currently executing SQL statements if there is a likelihood that one of these will be a re-trying deadlocked transaction).

Overall, I/Watch can help to make this difficult task much easier. In particular it will help to answer the most important question: whether deadlocking is caused by code design, CA-Ingres tuning, or a combination of both.

Dennis Adams is Senior Technical Consultant for Common Sense Computing (UK) Limited, which supplies Database Management and Monitoring Software for Ingres. He can be contacted via email on dennis@comsense.com.
