Informix Dynamic Server locking, Part 2: Tracking and analysis

Understand locking behavior and analyze lock conflicts in IDS

Eric Herber

January 04, 2007

Introduction

Multiuser databases, with many users and batch processes trying to update data simultaneously, must have mechanisms to protect the data and keep it consistent. Besides transaction logging, locking is the key feature that protects data integrity in the multiuser environment.

Locking often results in conflicts and wait situations, both common problems a DBA faces daily. And, analyzing locking problems without the appropriate scripts in place is complex and error-prone.

This article explains IDS locking mechanisms and helps you analyze lock conflicts and lock wait situations.

The examples are based on the stores_demo database, which can be created by executing the dbaccessdemo stores_demo -log command.

Dynamic lock allocation

How it works

The onconfig parameter LOCKS (max. 8.000.000) specifies the initial number of available locks for the whole instance. However, IDS comes with the dynamic lock allocation feature. If the initial number of locks has been reached, IDS automatically doubles the capacity of the internal lock table. This procedure is repeated up to 15 times. No more than 100.000 additional locks are allocated by each increase, so the maximum number of locks available in IDS is limited to 9.500.000:
8.000.000 initial + (15 x 100.000) additions = **9.500.000** total

The dynamic lock allocation feature allows an application to continue processing although the initial number of configured locks has already been reached.

**Suboptimal application code**

Sometimes, the dynamic allocation of locks *hides* suboptimal application code. A reasonable IDS feature request would be the ability to control the maximum number of locks that a single session is allowed to hold. This would help the DBA to identify and block badly written applications without affecting other database sessions.

Currently, if IDS 10 reaches the upper limit of 9.500.000 locks, all running applications receive a lock table overflow error when requesting a lock. This means all applications are affected by a problem created by a single, badly written application. This scenario is not ideal.

**Lock wait time**

**Settings per database session**

Each database session can set an individual lock wait time to prevent an overflow from affecting all applications. The IDS default is to set the lock mode to **not wait**, meaning as soon as a lock conflict is detected, IDS delivers these error codes to the application:

- `-244: Could not do a physical-order read to fetch next row`
- `107: ISAM error: record is locked`

You can specify the lock wait behavior per session with the following Informix extension to ANSI SQL:

- set lock mode to **not** wait
- set lock mode to **wait**
- set lock mode to **wait <#sec>**

**Distributed transactions**

For distributed transactions, the onconfig **DEADLOCK_TIMEOUT** parameter specifies the maximum amount of time that IDS will wait for a lock on a remote IDS instance before returning an error code to the application.

**Recommendation**

Specifying an unlimited lock wait (set lock mode to **wait**) is not the optimal solution. It increases the possibility of lock conflicts and deadlocks, as well as masking suboptimal application code.

Setting a lock wait time of 5-10 seconds is good practice in OLTP environments. If IDS is not able to acquire the lock in the specified time, an appropriate SQL error is delivered to the application. The application can now decide whether to retry the operation or rollback the current transaction.
Check session settings
You can check the current isolation level and the lock mode of database sessions with the `onstat -g sql` command:

**Listing 1. Session isolation level - onstat -g sql**

```plaintext
Output from the onstat -g sql command:

<table>
<thead>
<tr>
<th>Sess Id</th>
<th>SQL</th>
<th>Current Database</th>
<th>Iso Lock</th>
<th>SQL ERR</th>
<th>ISAM ERR</th>
<th>Vers</th>
<th>Explain</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>-</td>
<td>stores_demo</td>
<td>CRNot Wait</td>
<td>0</td>
<td>0</td>
<td>9.03</td>
<td>Off</td>
</tr>
<tr>
<td>16</td>
<td>-</td>
<td>stores_demo</td>
<td>RRWait 10</td>
<td>0</td>
<td>0</td>
<td>9.03</td>
<td>Off</td>
</tr>
</tbody>
</table>
```

This example shows two database sessions with different isolation levels:

- Session 18, **CR** Committed Read, **Not Wait**
- Session 16, **RR** Repeatable Read, **Wait 10**

Another reasonable IDS feature request would be the ability to set a default lock wait method to use if the application does not execute an explicit **set lock mode to wait...** statement on the database or instance level.

## Deadlocks
A deadlock occurs when two users hold locks, and each user wants to acquire a lock that the other user owns. To prevent deadlocks, IDS scans the internal lock table before granting new locks and delivers ISAM error code **-143 (deadlock detected)** to the affected database session.

### Number of deadlocks detected
The `onstat -p` command shows the total number of deadlocks (deadlks column) detected since IDS was started or since the `onstat -z` (zero statistics) command was executed. You can determine the last reset time of the IDS statistics using this SQL statement:

```sql
• select dbinfo("utc_to_datetime", sh_pfclrtime) from sysmaster:sysshmvals
```

### Sysmaster tables
Unfortunately, IDS does not provide many details about detected deadlocks and involved applications. This makes it difficult to analyze the cause of a deadlock after it's resolved. The sysmaster database contains some tables that offer additional information:

- **sysprofile**: Contains the total number of deadlocks detected which corresponds to the deadlks column in the `onstat -p` command output
- **syssesprof**: Contains the number of deadlocks detected per individual database session
- **sysptprof**: Contains the number of deadlocks detected per individual table

### Tracing deadlocks
The IDS error trapping feature is also useful in analyzing deadlocks. As already stated, IDS delivers ISAM error code **-143** to the application when detecting a possible deadlock. You can activate error trapping for deadlocks with the `onmode -I 143` command. IDS creates an assertion
failure file and stores the `onstat -a` output in this file as soon as a deadlock is detected. You can analyze the contents of this file and check the SQL statement that the deadlocked session was executing as well as the locks held by that session and other concurrent sessions.

**Deadlock timeout**

The `onconfig DEADLOCK_TIMEOUT` parameter has no influence on the deadlock detection time or the amount of time an application waits in the case of a local deadlock. IDS always immediately resolves local deadlocks.

If a lock wait occurs while selecting or updating data on a remote IDS instance, `DEADLOCK_TIMEOUT` specifies the upper limit of seconds that the local IDS instance will wait for the requested lock, after which it assumes that a deadlock may have occurred. ISAM error code `-154 (deadlock timeout expired - possible deadlock)` is then delivered to the application.

As stated, IDS does not crosscheck the internal lock table between instances; it uses the `DEADLOCK_TIMEOUT` parameter, which is more of a distributed lock timeout. The `DEADLOCK_TIMEOUT` parameter overwrites the application specific `set lock mode to wait` setting in case of distributed transactions.

**Analyze lock conflicts**

Often transactions try to access the same row concurrently in an OLTP environment with hundreds of parallel executing database sessions. Therefore, it is good practice to keep transactions as short as possible to avoid lock conflicts.

If you don't execute a `set lock mode to wait` statement in your application, you will receive an SQL error from IDS as soon as you request a lock on a row which is not compatible with the lock already placed on the same row from another session. A direct abort is often not desired, so applications normally execute the `set lock mode to wait` statement which instructs IDS to suspend the database session (sqlexec thread) until the lock can be granted or the agreed lock wait time has elapsed. However, this can lead to lock wait situations that can slow down the throughput of your IDS instance. As a result, interactive users might complain of slow response times or longer batch processes. Normally, you should not set the lock wait mode to "infinite" (see Recommendation).

Analyzing lock wait situations is a real challenge in a dynamic environment with ongoing transactions. The best method is to have some scripts in place that allow the real-time analysis of lock wait situations. The next section provides some helpful `onstat` commands. You can use these as a starter and write a more sophisticated script to automate the task, or you can take advantage of the `lockwt` utility.

**Useful onstat commands**

The `onstat -u` command is a good starter for looking after lock conflicts. There are two interesting values here:

- Number of locks currently **held** by each session
• The locks column shows you the number of locks held by this session. Sessions with a huge number here could be responsible for lock conflicts. This is not always the case, but a large number is probably an indicator of a poorly written application.

• Sessions currently waiting for a lock

• Database sessions waiting for a lock are marked with an 'L' in the first position of the flags column.

### Listing 2. Locks held and sessions waiting for a lock - onstat -u

Output from the onstat -u command:
```
address  flags  sessid   user     tty      wait     tout locks nreads nwrites
4506b44c L-BPR-- 20       informix 11       440cfac4 -1   17    19       0
4506b978 Y--P--D 16       informix -        4407d138  0    0     0        0
... ...
```

You can execute the onstat -k | grep 'L-' command to identify all sessions currently waiting for a lock. With onstat -g ses <sessid>, you can monitor which SQL statement this database session is currently executing. You can also see the opened database (Current Database), the current isolation level (Iso Lvl), and the agreed lock wait time (Lock Mode). In the status column, you can see the number of seconds left before IDS returns a lock timeout error to the application.

### Listing 3. Analyze sessions in lock wait status - onstat -g ses <sessid>

Output from the onstat -g ses 20 command:
```
42       sqlexec  4506b44c L-BPR--  7168     sleeping(secs: 9)
... ...
```

Next, identify the session that is holding the lock and causing the lock wait. The onstat -k command delivers an overview of all locks currently allocated in this IDS instance:

### Listing 4. Locks currently allocated in IDS - onstat -k

Output from the onstat -k command:
```
<table>
<thead>
<tr>
<th>address</th>
<th>wtlist</th>
<th>owner</th>
<th>lklist</th>
<th>type</th>
<th>tblsnum</th>
<th>rowid</th>
<th>key#/bsiz</th>
</tr>
</thead>
<tbody>
<tr>
<td>440cf9bc</td>
<td>0</td>
<td>4506b44c</td>
<td>0</td>
<td>HDR+S</td>
<td>100002</td>
<td>297</td>
<td>0</td>
</tr>
<tr>
<td>440cfa14</td>
<td>0</td>
<td>4506b44c</td>
<td>440cf0bc</td>
<td>HDR+IX</td>
<td>10015f</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>440cfa6c</td>
<td>0</td>
<td>4506b44c</td>
<td>440cfa14</td>
<td>HDR+X</td>
<td>10015f</td>
<td>101</td>
<td>0</td>
</tr>
<tr>
<td>440cfac4</td>
<td>4506b44c</td>
<td>0</td>
<td>440cfa6c</td>
<td>HDR+</td>
<td>10015f</td>
<td>106</td>
<td>0</td>
</tr>
<tr>
<td>440cfb1c</td>
<td>0</td>
<td>4506b44c</td>
<td>440cfa6c</td>
<td>HDR+X</td>
<td>100160</td>
<td>101</td>
<td>K-1</td>
</tr>
<tr>
<td>440cfb74</td>
<td>0</td>
<td>4506b44c</td>
<td>440cfb1c</td>
<td>HDR+X</td>
<td>100175</td>
<td>101</td>
<td>K-1</td>
</tr>
<tr>
<td>440cfbcc</td>
<td>0</td>
<td>4506b44c</td>
<td>440cfb74</td>
<td>HDR+X</td>
<td>10015f</td>
<td>102</td>
<td>0</td>
</tr>
</tbody>
</table>
... ...
```
The interesting column in this output is the second one, called \textit{wtlist}. It contains the hexadecimal address of the user thread waiting for a lock. However, depending on the number of locks currently active in an IDS instance, an \texttt{onstat -k} output could be quite large. You can use one of the following commands to locate the lock on which the database session with shared memory address \texttt{4506b44c} is currently waiting on:

\begin{itemize}
  \item AWK: \texttt{onstat -k | awk '$2 ~/4506b44c/ \{ print \}'}
  \item PERL: \texttt{onstat -k | perl -ane \texttt{\'print if $F[1] eq "4506b44c"\'}
\end{itemize}

The \texttt{owner} column gives you the shared memory address of the user thread holding the lock. Identifying this session is easy using the \texttt{onstat -u} command and piping the output to the \texttt{grep} utility. You can analyze what the cause of the lock wait is currently doing using the \texttt{onstat -g <sessid>} command. If the cause itself is waiting for a lock ('\texttt{L}' in the first position of the \texttt{flags} column in \texttt{onstat -u}), you have to repeat the steps mentioned above to find the cause of the second lock wait situation.

**Lockwt utility**

Using the Esql/C lockwt utility is a comfortable way to analyze lock wait situations. To use this utility, you need to install the Informix Client SDK and a C compiler in order to compile it. The lockwt utility searches through a pair of sysmaster tables to find lock wait situations.

The program reports each user session holding a lock and the session(s) waiting for this lock to be released. Execute the \texttt{lockwt -r <#sec>} command to repeat the lock-search in the specified time-interval (similarly to the \texttt{onstat -r} command).

The lockwt utility allows the real-time monitoring of complex lock wait situations, representing the collected information in an easy-to-read format.

**Listing 5. Lockwt - Description of the output format**

Output from lockwt:

\begin{verbatim}
(0) (1) (2) (3) (4) (5) (6) (7) (8) (9)
WAIT SID :PID  PROCNAME    USERNAME           LKTYPE    DATABASE:TABLENAME   LKOBJ
0 -   13900:12303 workprocess3   dbuser                  X        rome    :orders          row
1 W   53600:23613 batchp12       dbuser                           rome    :orders
\end{verbatim}

Colno Purpose

(0) Waiting or not waiting, possible values are:
  "-" - this session is the holder of the lock and is always listed first.
  "W" - this session(s) is(are) waiting for the above session.

(1) Session ID of this session in the database server

(2) Process ID of the UNIX process, remote connections have pid -1

(3) Process name of the UNIX process. If it is a remote connection (pid = -1), no process name will be available.

(4) UNIX username of this session
Type of lock, possible values are:
  "X" - Exclusive Lock
  "S" - Shared Lock
  "U" - Update Lock
For additional lock types, execute the following SQL statement:
-> select txt from sysmaster:flags_text where tabname = "syslcktab"

Table name, the lock is on. If it is an index lock and the index is detached
from the table (has its own partition number), the name of that index
is shown here.

Type of object locked, possible values are:
  "table" - this is a table lock
  "idx" - this is an index key lock
  "page" - this is a page lock
  "row" - this is a row lock

Listing 6. Lockwt - Lock wait situation I

Output from lockwt:
-------------------
<table>
<thead>
<tr>
<th>WAIT</th>
<th>SID</th>
<th>:PID</th>
<th>PROCNAME</th>
<th>USERNAME</th>
<th>LKTYPE</th>
<th>DATABASE:TABLENAME</th>
<th>LKOBJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13900:12303</td>
<td>workprocess3</td>
<td>dbuser</td>
<td>X</td>
<td>rome</td>
<td>:orders</td>
<td>row</td>
</tr>
<tr>
<td>1 W</td>
<td>53600:23613</td>
<td>batchp12</td>
<td>dbuser</td>
<td></td>
<td>rome</td>
<td>:orders</td>
<td></td>
</tr>
</tbody>
</table>

In this example session, 13900 (process "workprocess3") is holding a lock on a specific row in the
table orders. Session 53600 is waiting for this lock to be released. You need to analyze session
13900 by executing onstat -g ses 13900.

Listing 7. Lockwt - Lock wait situation II

Output from lockwt:
-------------------
<table>
<thead>
<tr>
<th>WAIT</th>
<th>SID</th>
<th>:PID</th>
<th>PROCNAME</th>
<th>USERNAME</th>
<th>LKTYPE</th>
<th>DATABASE:TABLENAME</th>
<th>LKOBJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 W</td>
<td>3894:</td>
<td>-1 (remote)</td>
<td>eherber1</td>
<td>X</td>
<td>rome</td>
<td>:status</td>
<td>row</td>
</tr>
<tr>
<td>1 W</td>
<td>17048:3140</td>
<td>batchp3</td>
<td>dbuser</td>
<td></td>
<td>rome</td>
<td>:status</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>63296:</td>
<td>-1 (remote)</td>
<td>eherber1</td>
<td>X</td>
<td>rome</td>
<td>:customer_order</td>
<td>row</td>
</tr>
<tr>
<td>1 W</td>
<td>3894:</td>
<td>-1 (remote)</td>
<td>eherber1</td>
<td></td>
<td>rome</td>
<td>:customer_order</td>
<td></td>
</tr>
</tbody>
</table>

This example is a little bit more complex. Session 17048 is waiting for session 3894 to release
the lock on table status. But take a look at the second pair of locks. Session 3894 is waiting for
session 63296. This is a typical escalating lock situation, because session 3894 is holding a lock
another session is waiting for, but session 3894 is also waiting for a lock to be released. Analyze
what session 63296 is doing using the onstat -g ses 63296 command.

In the source code of the lockwt utility, there are some interesting sysmaster queries that you can
take as a starter for writing specific queries on your own.

The open cursor problem

You may have already encountered a strange problem when trying to alter a table. Even though
you have explicitly locked a table in exclusive mode, you are not able to alter it. The following
example demonstrates this issue:
Listing 8. Non-exclusive access on a table

Output from dbaccess -e stores_demo <script.sql>:
--------------------------------------------------
begin;
    Started transaction.
lock table customer in exclusive mode;
    Table locked.
alter table customer add (mycol integer);
    242: Could not open database table (informix.customer).
    106: ISAM error: non-exclusive access.
This behavior occurs because a select cursor was opened on table customer by someone else. The cursor is not placing any locks on individual rows; otherwise we would have not been able to lock the table exclusively, but it prevents IDS from changing the partition information.

To solve the problem, identify the session that opened a cursor on table customer:

1. Determine the hexadecimal partition number of table customer:
   - Select hex(partnum) from systables where tabname = "customer".
2. If the partition number of this table is zero, it is a fragmented table. You need to execute the following SQL statement ito find the partition numbers of the individual fragments:
   Select st.tabname, dbinfo("dbspace", sf.partn), hex(sf.partn) from systables st, sysfragments sf, where st.tabid = sf.tabid and sf.fragtype = "T"and st.tabname = "customer".
3. Take the hexadecimal partition number and search for it in all currently opened tables:
   onstat -g opn | grep -i <hex_partnum>
4. Take the rstcb column, which is a shared memory address of the respective user thread, and search for it using the onstat -u command.
   onstat -u | grep <rstcb_without_leading_0x>

After you identify the respective database session, you can terminate it with the onmode -z <sessid> command.

If you're running on IDS version 7.31.xD5, 9.40 or 10, you can also take advantage of the environment variable IFX_DIRTY_WAIT. This environment variable can be set in the engine or in the client environment. IFX_DIRTY_WAIT specifies the number of seconds that a DDL statement will wait for existing dirty readers to finish accessing the table that is about to be altered. If the specified number of seconds has expired, IDS returns the same error to the application that would be sent if the IFX_DIRTY_WAIT variable was not set.

Summary

Analyzing lock conflicts in real-time, with hundreds of parallel executing transactions, is a challenge. But hopefully this article explained IDS locking mechanisms concisely, and will make solving lock conflicts less time-consuming.
As I write this article, the next version of IDS (IBM's preferred data server for high-end OLTP and embedded environments), is entering the beta phase. The new version will offer interesting, new options regarding locking. Stay tuned!
Related topics

- Download other Informix evaluation and trial software.
- IDS 10 Information Center: Get easy Web-based access to the whole documentation set of IDS Version 10. for more information on IDS 10.
- Visit The Informix Zone: Where technology meets marketing
- developerWorks Informix zone: Read articles and tutorials, and connect to other resources to expand your Informix skills.
- Build your next development project with IBM trial software, available for download directly from developerWorks.

© Copyright IBM Corporation 2007
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)