Bugs - Bug #1455

P2J server virtual session deadlock

07/25/2012 05:03 PM - Eric Faulhaber

Status:	Closed	**Start date:**	07/03/2012
Priority:	High	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

P2J server cannot create new virtual session due to a deadlock within RouterSessionManager.

History

#1 - 07/25/2012 05:06 PM - Eric Faulhaber

Email Discussion from 2012-07-03

Eric,

The Reader/Writer threads for the remote database connection don't have an explicit name, as the threads for the client Conversation thread have. So,the Reader thread belongs to the server to server connection; I agree the session may not be the same, but what we do know is that both are virtual sessions and both use the same queue (as the queue object is part of the deadlock).

Is there a chance that you instantiate a buffer for the remote database after it disconnected? At this time I can't imagine how a session can reestablish connection after the queue had initiated its shutdown... Can I take a look at the QA Vendor code? Also, let me know if you want me to digg more... at this time what I am almost sure of is that the race condition happens during remote queue shutdown.

Thanks,
Constantin

Eric Faulhaber <ecf@goldencode.com> wrote:

> Constantin,
>
> Thanks for looking at this. Please see answers below.
>
> On 07/03/2012 04:36 AM, Constantin Asofiei wrote:
>
> > Eric,
> >
> > I think the problem is not the RSM.connectVirtual; the problems I see are in the virtual session cleanup/disconnect code.Â The RSM.connectVirtual block you saw in the thread dump is just a side-effect of Reader (for the server-to-server connection) and Conversation threads deadlocking while cleaning up and disconnecting the virtual session (see the last two threads in the thread dump).
> >
> > The real problem is that both Reader and Conversation threads reach the session.terminate() call in the DirtyShareFactory.unregisterManagerForDatabase method call (for the same remote database), while cleaning up and disconnecting the remote database.Â I don't know how this can be possible, as the DSF.unregisterManagerForDatabase is sync'ed on the DSF.cache object, and before session.terminate() is called, the session object is removed from the DSF.sessions map (thus should be used by only one of the threads).
> >
> > I don't think we can assume it is the same Session object in both threads. I'm pretty sure I started and ended multiple sessions (in sequence, not in parallel) before hitting this deadlock. So, the deadlocked conversation and reader threads are not necessarily from the same session. Note that the reader thread name does not yet include a context ID and user ID, which suggests it is terminating before it was fully initialized.
> >
> > I think some other thread manages to call DSF.getManagerInstance (and inject a session object in the DSF.session map) for the remote database while the queue is shutting down.Â Can you give me a scenario on how you use the virtual sessions on the QA vendor? Have you made any changes in the DirtyShareFactory class?
>
> I call ConnectionManager.connect in a loop to connect multiple sites at once:

```
...
ConnectionManager.connect(DB_NAME,
ldbName,
scd.getHost(),
socket);
...
```

I haven't changed DirtyShareFactory. The only P2J runtime change I've made that is not checked in is an additional API to the ConnectionManager, to allow me to get a Persistence object from hand-written Java code, in order to perform queries:

```
   /**
    * Get the {@link Persistence} instance associated with the given database
    * name, which may be a physical or logical name, or an alias.
    *
    * @param   name
    *          Physical or logical database name, or alias.
    *
    * @return  Persistence object associated with the given database, which
    *          may be locally or remotely connected.
    */
   public static Persistence getPersistence(String name)
   {
      Database database = get().getDatabase(name);

      return PersistenceFactory.getInstance(database);
   }
```

This is only called after the remote connections have been made. I don't make any other calls to any SessionManager instance or to DirtyShareFactory directly.

Thanks,
Eric

> Thanks,
> Constantin
>
> On 07/03/2012 07:08 AM, Eric Faulhaber wrote:
>
> > Constantin,
> >
> > This is a server deadlock that has occurred on my system with some as yet unpublished QA vendor changes.Â It seems to be a race condition triggered by hand-written code which calls RouterSessionManager.connectVirtual on multiple, remote servers from the same session.Â So, it's definitely a new use case that's triggering it, but it's one we will have to support in the very near future.Â It doesn't occur that frequently;Â this is the second time I've seen it in a few weeks.
> >
> > Although you don't have the code to recreate it yet, hopefully the stack trace and deadlock info can give you a good idea what's going on.Â Please let me know if you think this will be painful/fragile to fix.Â The fact that RSM.connectVirtual is itself synchronized and then internally synchronizes on an inherited lock looks suspicious at a glance, but I haven't looked deeply at it.
> >
> > Thanks,
> > Eric

#2 - 07/25/2012 05:18 PM - Eric Faulhaber

Further Email Discussion

Eric,

I've managed to duplicate your error ... while in the "SPECIAL QA SCREEN" screen (V/F/S), if you disconnect the client abruptly (i.e. just close the terminal window), next time when you try to connect, the application blocks. The deadlock I think happens when the client ends abruptly, and is visible only on next remote DB connection attempt. The recreate is stable, I've tested it a few times, and each time it deadlocked.

Eric: do you remember doing something like this (closing the client window directly) in your tests ?

PS: the files in /gc/convert/staging_qavendor/ are owned by ECF and not writable by group, so I'm working in a copy, in staging_ca/qa/

Thanks,
Constantin

On 07/25/2012 11:32 AM, Eric Faulhaber wrote:

> Constantin,
>
> I've finally gotten the QA Vendor update which triggered this problem configured and working on lightning, in /gc/convert/staging_qavendor. You can connect using a client with -i17. This will connect you with a GSO server (there are also LCQ and MCN servers running).
>
> As a matter of background information, the feature set is documented under issue #1677 on TIMCO's Redmine system. I have not yet had a chance to put together an issue on the P2J defects in GCD's Redmine system, and right now, I'm just too tired. I will do this later today after some sleep, or perhaps you can, if you get anywhere with this issue today.
>
> In case you need the source from git, my updates are pushed to the majic.git repo on lightning, in branch RM_1677_20120724. However, the p2j subdirectory of /gc/convert/staging_qavendor is taken from /gc/convert/staging (with minor patches applied for this issue), since the version of p2j in git was out of date, at an unknown level.
>
> With the systems running on lightning, you should have enough to get started looking into it: the primary issue is the one described in our July 3 email discussion with the subject "RouterSessionManager deadlock". You should already have the thread dump from that failure.
>
> IIRC, the failure manifested itself after I left the "Special QA Screen" (F, V, S from the main Majic menu), when I was trying to get back into that screen. I have done this successfully many times, but there seems to be something about the timing which makes it fail occasionally. Having dug through the thread dump previously, you may be able to set breakpoints to recreate the problem. I have started all 3 servers with the -d option. See the TIMCO RM 1677 entry for server startup details, in case you need to restart them.
>
> I encountered another problem more recently (perhaps with a similar root cause), which again got the server into a state where it could not create a new virtual session. But I have to play with the system here a bit more to see if I can recreate it and provide more detail.
>
> Thanks,
> Eric

#3 - 07/25/2012 05:26 PM - Eric Faulhaber

Thanks for the instructions; I have been able to recreate the lockup using your scenario. But, I'm pretty sure an abrupt client termination did not happen on the few occasions where I saw this during my development, so there is probably another, less dramatic way this can occur as a race condition.

BTW, I get a ConcurrentModificationException in DatabaseManager code when I kill the client:

```
[07/25/2012 16:51:08 EDT] (TransactionManager.handleDeferredError:SEVERE) {00000001:00000009:syman} <depth = 1
1; trans_level = -1; trans_label = null; rollback_scope = -1; rollback_label = null; rollback_pending = false;
 in_quit = false; retry_scope = -1; retry_label = null; ignore_err = false> [label = outerLoop; type = REPEAT;
 full = false; trans_level = SUB_TRANSACTION; external = false; top_level = false; loop = true; loop_protectio
n = false; had_pause = false; endkey_retry = false; next_or_leave = leave; is_retry = false; needs_retry = fal
se; ilp_count = -1; pending_break = false; properties = 'ERROR, ENDKEY'] Throwing deferred error (Connection e
nded abnormally)
[07/25/2012 16:51:08 EDT] (SecurityManager:SEVERE) {00000001:00000009:syman} Error cleaning up security contex
t
java.util.ConcurrentModificationException
        at java.util.HashMap$HashIterator.nextEntry(HashMap.java:793)
        at java.util.HashMap$KeyIterator.next(HashMap.java:828)
        at com.goldencode.p2j.persist.DatabaseManager$1.cleanup(DatabaseManager.java:373)
        at com.goldencode.p2j.persist.DatabaseManager$1.cleanup(DatabaseManager.java:356)
        at com.goldencode.p2j.security.ContextLocal$Wrapper.cleanup(ContextLocal.java:358)
        at com.goldencode.p2j.security.SecurityContext.cleanup(SecurityContext.java:419)
        at com.goldencode.p2j.security.SecurityManager.endContext(SecurityManager.java:6664)
        at com.goldencode.p2j.security.SecurityManager.popContextWorker(SecurityManager.java:6638)
        at com.goldencode.p2j.security.SecurityManager.popAndRestoreSecurityContext(SecurityManager.java:3875)
        at com.goldencode.p2j.net.RouterSessionManager.restoreContext(RouterSessionManager.java:1041)
        at com.goldencode.p2j.net.Conversation.run(Conversation.java:191)
        at java.lang.Thread.run(Thread.java:662)
```

I don't know if this is a cause, or just a secondary issue. Please continue to investigate and document what you find.

#4 - 07/26/2012 10:44 AM - Constantin Asofiei

Looking at the ConcurrentModificationException above, at the deadlock in the attached thread dump and at the deadlock which happens when the client is killed:

```
"Reader":
    at com.goldencode.p2j.net.SessionManager.terminateVirtual(SessionManager.java:363)
    - waiting to lock <0x0000002abef7a568> (a com.goldencode.p2j.net.RouterSessionManager)
```

```
    at com.goldencode.p2j.net.RequesterSession.terminate(RequesterSession.java:93)
    at com.goldencode.p2j.persist.ConnectionManager.disconnectImmediately(ConnectionManager.java:1575)
    at com.goldencode.p2j.persist.ConnectionManager.disconnectImmediately(ConnectionManager.java:1547)
    at com.goldencode.p2j.persist.ConnectionManager.sessionClosed(ConnectionManager.java:1529)
    at com.goldencode.p2j.persist.ConnectionManager.access$100(ConnectionManager.java:140)
    at com.goldencode.p2j.persist.ConnectionManager$1.cleanup(ConnectionManager.java:156)
    at com.goldencode.p2j.persist.ConnectionManager$1.cleanup(ConnectionManager.java:148)
    at com.goldencode.p2j.security.ContextLocal$Wrapper.cleanup(ContextLocal.java:358)
    at com.goldencode.p2j.security.SecurityContext.cleanup(SecurityContext.java:419)
    at com.goldencode.p2j.security.SecurityManager.endContext(SecurityManager.java:6664)
    at com.goldencode.p2j.security.SecurityManager.popContextWorker(SecurityManager.java:6638)
    at com.goldencode.p2j.security.SecurityManager.popAndRestoreSecurityContext(SecurityManager.java:3875)
    at com.goldencode.p2j.net.RouterSessionManager.restoreContext(RouterSessionManager.java:1041)
    at com.goldencode.p2j.net.Queue.stop(Queue.java:414)
    - locked <0x0000002ac4cdc5d0> (a com.goldencode.p2j.net.Queue)
    at com.goldencode.p2j.net.Protocol$Reader.run(Protocol.java:416)
    - locked <0x0000002b73d07f10> (a java.lang.Object)
    at java.lang.Thread.run(Thread.java:619)
"Conversation [00000001:syman]":
    at com.goldencode.p2j.net.SessionManager.deregisterSession(SessionManager.java:1111)
    - waiting to lock <0x0000002ac4cdc5d0> (a com.goldencode.p2j.net.Queue)
    at com.goldencode.p2j.net.RouterSessionManager.deregisterSession(RouterSessionManager.java:885)
    at com.goldencode.p2j.net.SessionManager.endSession(SessionManager.java:981)
    at com.goldencode.p2j.net.SessionManager.terminateVirtual(SessionManager.java:376)
    - locked <0x0000002b73eb75f0> (a java.lang.Object)
    - locked <0x0000002abef7a568> (a com.goldencode.p2j.net.RouterSessionManager)
    at com.goldencode.p2j.net.RequesterSession.terminate(RequesterSession.java:93)
    at com.goldencode.p2j.persist.ConnectionManager.disconnectImmediately(ConnectionManager.java:1575)
    at com.goldencode.p2j.persist.ConnectionManager.disconnectImmediately(ConnectionManager.java:1547)
    at com.goldencode.p2j.persist.ConnectionManager.sessionClosed(ConnectionManager.java:1529)
    at com.goldencode.p2j.persist.ConnectionManager.access$100(ConnectionManager.java:140)
    at com.goldencode.p2j.persist.ConnectionManager$1.cleanup(ConnectionManager.java:156)
    at com.goldencode.p2j.persist.ConnectionManager$1.cleanup(ConnectionManager.java:148)
    at com.goldencode.p2j.security.ContextLocal$Wrapper.cleanup(ContextLocal.java:358)
    at com.goldencode.p2j.security.SecurityContext.cleanup(SecurityContext.java:419)
    at com.goldencode.p2j.security.SecurityManager.endContext(SecurityManager.java:6664)
    at com.goldencode.p2j.security.SecurityManager.popContextWorker(SecurityManager.java:6638)
    at com.goldencode.p2j.security.SecurityManager.popAndRestoreSecurityContext(SecurityManager.java:3875)
    at com.goldencode.p2j.net.RouterSessionManager.restoreContext(RouterSessionManager.java:1041)
    at com.goldencode.p2j.net.Conversation.run(Conversation.java:191)
    at java.lang.Thread.run(Thread.java:619)
```

I think the root cause is related to the context cleanup code. The ConcurrentModificationException happens because two threads (which are authenticated in the same context) execute the context cleanup code simultaneously. In these cases, these threads are a Conversation thread and a Reader thread associated with a virtual session. When the Conversation thread initiates context cleanup, it sends commands to terminate the remote side of the virtual session. After this, the Reader thread associated with the virtual session receives the command to end, executes the code to stop the queue, and finally it needs to clean up the context's resources. But, as the Conversation thread is in the middle of cleaning up these resources too, problems appear.

A solution is to "bullet proof" the context cleanup methods in DatabaseManager and ConnectionManager, so that only one thread could execute the cleanup code (i.e. add some "cleaningUp" flag which blocks other threads to execute the cleanup code, if another one started it). But I don't think this is the right solution...

Greg: was the SecurityManager.endContext (for a certain security context) method meant to be executed only from a single thread at a time, and concurrency was never in mind ? If yes, then the problem is at the SecurityManager level, and we should fix it there.

#5 - 07/26/2012 03:53 PM - Greg Shah

- Project changed from Core Development to Bugs

#6 - 07/26/2012 04:01 PM - Greg Shah

- Status changed from New to WIP

```
But I don't think this is the right solution...
```

Yes, you're right.

```
was the SecurityManager.endContext (for a certain security context) method meant to be executed only from a si
ngle thread at a time, and concurrency was never in mind ?
```

Yes, that is exactly the idea. I suspect we should have synchronized on the context object itself.

```
If yes, then the problem is at the SecurityManager level, and we should fix it there.
```

Absolutely right.

#7 - 07/27/2012 08:33 AM - Constantin Asofiei

The following files in com.goldencode.p2j.security have been changed for this issue:

1. SecurityManager.endContext - only the first thread which reaches the context cleanup code will execute it; if other threads get here during context cleanup, they will no-op.

1. SecurityContext.cleanup - it fixes a potential problem, not related to #1455. Idea is, looking for Cleanable only through the directly implemented interfaces poses a problem, if the Cleanable is implemented by a superclass or someone else up the hierarchy. Instead, Class.isAssignableFrom must be used.

#8 - 07/27/2012 08:37 AM - Constantin Asofiei

- Status changed from WIP to Review

#9 - 07/27/2012 10:33 AM - Greg Shah

- Status changed from Review to WIP

I have 1 question and 1 concern:

Question: the simpler approach seems to be synchronizing on the SecurityContext instance itself (and making any other conflicting threads wait until cleanup is done). Why not take that approach?

Concern: I think there is still a possibility that 2 threads (or more?) will call SecurityContext.cleanup(). You have ensured that they can never call it at the same time. But those threads may call endContext() sequentially (without hitting your protection logic). I think we need protection logic in SecurityContext.cleanup() to ensure that it only is ever called once.

#10 - 07/27/2012 11:28 AM - Constantin Asofiei

Greg, good notes.

> Question: the simpler approach seems to be synchronizing on the SecurityContext instance itself (and making any other conflicting threads wait until cleanup is done). Why not take that approach?

This will not work because it will deadlock. Consider this scenario: Conversation thread starts the context cleanup code (acquiring a lock on the SecurityContext instance). The cleanup code for ConnectionManager will send the command to end the virtual session and will wait for this command to complete. When the Reader thread for the virtual session notices that it needs to end, it will stop the queue and will AGAIN start the context cleanup code for the same SecurityContext instance (as the Reader for the virtual session is closed using the context of the initiator). As the Conversation thread has a lock on this, Reader can not go forward until Conversation thread releases the lock. But Conversation thread is still in waiting mode to receive an answer for its "end virtual session" command... so, deadlock.
Just to note, I've already tried this approach and deadlocked was confirmed.

> Concern: I think there is still a possibility that 2 threads (or more?) will call SecurityContext.cleanup(). You have ensured that they can never call it at the same time. But those threads may call endContext() sequentially (without hitting your protection logic). I think we need protection logic in SecurityContext.cleanup() to ensure that it only is ever called once.

All Cleanable instances are kept in SecurityContext.tokenMap, and a copy of this map will be iterated when endContext() is called. But, after Cleanable.cleanup is called, the instance will be removed from the tokenMap. So, a subsequent call (from another thread) will be a no-op, as tokenMap will be empty.
Why I say this: the cleanup code for a tokenMap entry is not implemented in a Cleanable.cleanup method, but in a ContextLocal.cleanup method. Also, the ContextLocal.cleanup code is invoked via a ContextLocal.Wrapper.cleanup call (the Wrapper being the actual Cleanable instance which is used in SecurityContext.cleanup), and after it calls the ContextLocal.cleanup() method, it calls security.removeToken(ContextLocal.this) (see ContextLocal.Wrapper.cleanup():362), which removes the Cleanable instance from the tokenMap.

My conclusion is that the endContext code will always remove all Cleanable instances from the tokenMap, so protection is not needed at this time (as if any future thread will call it again for same context, it will find an empty tokenMap). But, if you think tokenMap can end up with Cleanable instances which are not automatically removed from tokenMap after cleanup() code is executed, then we should add that protection. Although if Cleanable is not removed from tokenMap will lead to mem leaks...

#11 - 07/27/2012 11:48 AM - Greg Shah

1. Please update the SecurityManager.endContext() code with an explanation of the deadlock (including your example) with a warning that simple synchronization on the SecurityContext instance cannot be used there.

2. Thanks for the explanation. I see that in the common case (and probably all of our current use cases), this problem is not an issue. However, the design is hard to see (by reading the code). In addition, the design is not foolproof (against mistakes or malicious intent). The problem I see is any case where someone has inserted an instance that is not based on the ContextLocal.Wrapper class. This can easily be done directly with the security manager. Yes, we do use ContextLocal for all of our P2J stuff. But future developers may not know that or may decide to do it differently. We don't want the clearing of the SecurityContext.tokenMap to be dependent upon a single outside class' implementation. In addition, any objects in the tokenMap that are not Cleanable are left there. Although if the SecurityContext instance is garbage collected, that won't be a problem, yet I still think we should be on the safe side. Please update the SecurityContext.cleanup() method to go ahead and do the following:
- Can we make the method synchronized? That way it is safe to modify the map and we know we are the only thread there.
- Let's explicitly clear the tokenMap at the end of the method.
- Any subsequent calls will find the map empty and the method will quickly return.

#12 - 07/30/2012 06:15 AM - Constantin Asofiei

For #2 - we can't make the method synchronized, for the same reason why SecurityManager.endContext can't sync on the SecurityContext instance: when the Reader (for the virtual session) wants to stop the queue, it needs to use the current context, so that the queue stop can be executed: but, calling SecurityContext.assign needs a lock on the SecurityContext instance, which is already acquired by the Conversation thread. So, deadlock.

An alternative would be to clear the tokenMap at the start of the SecurityContext.cleanup method (just after it is cloned). This way, no other thread would be able to iterate it.

#13 - 07/30/2012 08:13 AM - Greg Shah

OK, I understand about the inability to synchronize that method. Please place comments there describing the problem.

Go ahead with the idea to clear the map at the start of the cleanup() method. I assume it would be done while in the synchronized block.

#14 - 07/30/2012 10:23 AM - Greg Shah

- Status changed from WIP to Test

I approve the latest proposed code changes. Please apply the changes to lightning staging and run regression testing there. If it passes, you may check it in and distribute the update.

When that happens, Eric will promote the build so that the "pending" will have all the required fixes.

#15 - 07/31/2012 03:12 PM - Constantin Asofiei

The staging has passed regression testing (both standard and CTRL-C sets).

#16 - 10/24/2012 10:29 AM - Greg Shah

- Status changed from Test to Closed

#17 - 10/26/2012 12:53 PM - Greg Shah

- File ca_upd20120727a.zip added

- File ca_upd20120730a.zip added

Files

threaddump-1341287442432.tdump	67.6 KB	07/25/2012	Eric Faulhaber
ca_upd20120727a.zip	51.8 KB	10/26/2012	Greg Shah
ca_upd20120730a.zip	52.5 KB	10/26/2012	Greg Shah