Database - Bug #4796

locking regression in RAQ introduced with the FFC

07/20/2020 01:28 PM - Ovidiu Maxiniuc

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Database - Feature #4033: avoid query for FIND unique when the cur...	**WIP**

History

#1 - 07/20/2020 01:34 PM - Ovidiu Maxiniuc

Initial issue:

When we have a FF cache hit, we do not honor the record lock request of the FIND. So, a request to FIND ... EXCLUSIVE-LOCK, for instance, will not apply the EXCLUSIVE-LOCK on a cache hit.

Initially proposed solution:

Similarly to how we acquire/release the lock in Persistence.load, implement the locking using RecordLockContext for the cache hit case.

Current status:

I analysed the executeImpl() method which was shortcut when the ff cache returns with success. I copied the area which handles the locking, but I am trying to avoid code duplication using some lambda expressions, specific to each caller. I can commit this in a couple of hours, after testing it a bit.

However, after handing the locking, I noticed that the possible result from executeImpl() method is then checked against the dirty share manager for an overriding result. I am trying to decide whether this needs to be done after the ff cache hit. It's a bit complicated since the dirty share is not fully functional yet. Since it is disabled for some customer projects I am inclined to ship this for the moment.

#2 - 07/20/2020 02:26 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I copied the area which handles the locking, but I am trying to avoid code duplication using some lambda expressions, specific to each caller.

I am not sure what this means. The locking is pretty tightly integrated into Persistence.load logic. Currently, there is only one caller of this method

(RAQ.executeImpl). It is public, so conceivably there could be more in hand-written application code, but that is unlikely, since RecordBuffer instances are not meant to be used outside the persist package.

> However, after handing the locking, I noticed that the possible result from executeImpl() method is then checked against the dirty share manager for an overriding result. I am trying to decide whether this needs to be done after the ff cache hit. It's a bit complicated since the dirty share is not fully functional yet. Since it is disabled for some customer projects I am inclined to ship this for the moment.

Can you think of any case where we would need to consult the dirty share manager for a different result than what we got from the FFC? If the dirty share manager contained different state, whatever caused that change in state should have invalidated the cache, no?

As I think more about this, we may need to use the cache result only for NO-LOCK requests, or where we already have at least a SHARE-LOCK on the cached record. Persistence.load guarantees (or at least it used to) that we have the requested lock type before fetching the record data. We don't know we have the latest record state if the primary key comes from the cache.

#3 - 07/20/2020 02:50 PM - Eric Faulhaber

Eric Faulhaber wrote:

> As I think more about this, we may need to use the cache result only for NO-LOCK requests, or where we already have at least a SHARE-LOCK on the cached record. Persistence.load guarantees (or at least it used to) that we have the requested lock type before fetching the record data. We don't know we have the latest record state if the primary key comes from the cache.

Here's a proposed approach for handling the FFC with requests for FIND with a lock:

1. check the cache; if a miss, continue as normal (i.e., run executeImpl);
2. if we get a hit, get the requested lock with RecordLockContext;
3. check whether the result still exists in the cache, now that we hold the lock;
4. if it does not, someone else made an invalidating change; continue as if we had a cache miss (i.e., run executeImpl);
5. if the result is still in the cache, it tells us no one changed the index(es) while we were acquiring the lock; it is safe to use;
6. finalize the find.

Do you see any holes in this approach?

#4 - 07/20/2020 02:50 PM - Eric Faulhaber

- Assignee set to Ovidiu Maxiniuc

- Status changed from New to WIP

#5 - 07/20/2020 03:02 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Do you see any holes in this approach?

No. In fact this is even simpler and straightforward than the solution I initially wrote.

#6 - 07/20/2020 03:13 PM - Eric Faulhaber

It is unfortunate that the RecordIdentifier used in the FF cache and session cache is not the same as can be used for locking. We will need a quick way to get the locking kind (i.e., table name), given an instance of the caching kind (i.e., DMO type name).

I think we need to treat a LockUnavailableException (for a NO-WAIT lock request) thrown when trying to acquire a lock on a cache result, as if it were a cache miss. It means someone else holds the lock on the cached result and the result is likely to be invalid when that lock is released.

#7 - 07/20/2020 05:15 PM - Ovidiu Maxiniuc

I committed revision 11579. I am unable to test at this moment with customer application.

#8 - 07/21/2020 02:13 AM - Eric Faulhaber

Code review 4011b/11579:

I had to make one change to get the RecordLockContext directly from RecordBuffer, rather than get the Persistence$Context instance first. This caused a problem in the event the DataModelObject passed to FindQuery was a proxy to a RecordBuffer proxy, rather than the concrete RecordBuffer instance. This is a less common case, and I think the problem is in the proxy code, but I did hit it while testing.

Note that the updateLock parameter is not being honored in the FFC case. We are updating the lock state unconditionally, even when updateLock is false.

The only other issue I see is that we are potentially leaking locks. We need to properly handle the case where we acquire the lock on the first cached result, but then the second cached result does not match the first, after we get the lock. In this case, we need to reset the lock back to what it was after executeImpl, *but only if the ultimate result of executeImpl is not the same record that we locked*.

#9 - 07/21/2020 02:37 AM - Eric Faulhaber

Eric Faulhaber wrote:

> I had to make one change ...

This was committed as rev 11580.

#10 - 07/22/2020 02:47 PM - Eric Faulhaber

- Related to Feature #4033: avoid query for FIND unique when the current record in the buffer would match the query result added