Database - Bug #7358

Optimize Session.invalidateRecords

05/17/2023 03:43 AM - Alexandru Lungu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Alexandru Lungu	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 05/17/2023 05:24 AM - Alexandru Lungu

I see Session.invalidateRecords as a constant bottle-neck in my profiling. Most of the times, TemporaryBuffer triggers this invalidation due to multiplex scope close or table clear (due to table parameter output copy, empty-temp-table or copy-temp-table).

The implementation is traversing the whole cache in the search of the records which should be removed. As the cache has 1024 records, this is inefficient and surely disallows us to increase the cache size without firing back. There are two issues here:

- The records are not filtered by the table. There is no quick way to find the records which should be pruned.
- The records store the multiplex in the value, not the key. Therefore, we should extract it from the value at each step as we don't have access to it in the key.

The operations we use on the session cache are:

- pufIfAbsent on associate
- keys to deregister from versioning
- remove on evict
- get on cache look-up
- cacheExpired on entry expiry

All of these are intensively used, so we shouldn't spoil the performance here. We need something like bulk identify (**not remove - allow us to independently do the remove of each cache entry**).

My initial idea here is to extent the cache with an additional WeakHashMap, much like the ffCache does with ReverseLookup. However, I don't have a clear idea of how the RecordIdentifier can help us here as a WeakHashMap key, when we are in need of a segregated table / multiplex / id key.

Therefore, I suggest managing a 2-level cache: Map<T, Map<Integer, Set<RecordIdentifier<T>> > > to help us do the pruning:

- The putIfAbsent will work by creating the required table entry / multiplex entry / recid entry.
- The keys and get won't affect this mapping
- The remove will need to do the proper remove here and clear-up. If the table / multiplex / recid map is empty, remove the entry from the parent map.
- The cache expiry will also trigger a remove in the mapping.

The prune is basically an in-depth get on this second map. The down-fall here is the need of creating one HashMap per table and one HashSet per multiplex of a table. I also expect a lot of trashing by removing / adding maps here.

Please let me know if you have other suggestions that can be applied here!

#2 - 05/19/2023 05:43 AM - Alexandru Lungu

- % Done changed from 0 to 100

- Status changed from WIP to Review

Committed implementation of the idea from #7358-1 to 7026d as rev. 14571.
The total time improvement is of **-5.2%** on a customer application POC. As this is a really big improvement rate, I would like a second pair of eyes here. It looks too good to be true :)

#3 - 05/19/2023 09:08 PM - Ovidiu Maxiniuc

I have reviewed revision 14571 of 7026d. I totally agree with the code. Good job!

If there is space for more improvement here is to change the RecordIdentifier type from String to Class<?>. In all cases of RecordIdentifier object creation as keys for the cache, the new RecordIdentifier<>(dmoImplClass.getName(), id) (or similar) is used. There would be two places for improvements:

1. dropping the extra .getName() calls and the strings do not need to be eventually interned (although I think they already are in this case);
2. the .class is an identifier so the much faster IdentityHashMap could be used (the calls to .equals() are replaced by ==).

Eric,
I remember we already have discussed choosing between String to Class<?>, but I do not remember why we stick to the former. Using the class name allows uncoupling the content of the cache from the DMO class which would allow the class to be unloaded (for dynamic temp-tables). However, we must evict the records first from the cache before DMO class unloading or else they will age forever, leaking memory.

#4 - 05/23/2023 10:31 AM - Eric Faulhaber

Ovidiu, you are right: I don't want to use Class<?> instead of String with RecordIdentifier. We need to eliminate strong references to DMO classes (I mean this in the general sense, including interfaces), so transient DMO classes can be unloaded when they are no longer in use. We already have strong references to these in numerous places (e.g., in the GCD proxy implementation, where DMO interfaces are stored in the cache keys). I don't want to add more; it's already a mess to untangle what we have.

#5 - 05/29/2023 05:31 AM - Alexandru Lungu

- Status changed from Review to Test

Merged 7026d to trunk as rev. 14587.

Eric Faulhaber wrote:

> Ovidiu, you are right: I don't want to use Class<?> instead of String with RecordIdentifier. We need to eliminate strong references to DMO classes (I mean this in the general sense, including interfaces), so transient DMO classes can be unloaded when they are no longer in use. We already have strong references to these in numerous places (e.g., in the GCD proxy implementation, where DMO interfaces are stored in the cache keys). I don't want to add more; it's already a mess to untangle what we have.

I see that the largest issue here is related to Class<?> being loaded and unloaded so that we always need to do this kind of clean-up (in caches which map by Class<?>). While I understand the reasoning, the performance of IdentityHashMap and == over Class<?> is considerable versus String.equals (unless interned).

My point is: can we save some performance here by using WeakIdentityHashMap? This is not only about this specific issue, but the "numerous places" you mention. Having such leak-proof solution which can also guarantee some performance through == seems nice. In the same time, it feels

more natural to work with the "type safe" Class<?>, so this is a second argument.

Out of context: I've seen H2 using SoftReference a lot for their caches. Maybe we should consider using such solution for our caches to avoid OOM for the cost of performance. This is also a very nice leak guard we can use.

Please let me know if we can close #7358 and move this discussion on a separate thread.