Database - Bug #7448

Optimize FWD-H2 ValueTimestampTimeZone and maybe avoid caching

06/19/2023 09:09 AM - Alexandru Lungu

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

Description

Related issues:

Related to Database - Feature #7363: Improve H2 Value caching, hashing and eq...	**Test**
Related to Database - Bug #7452: Analyze memory of the value cache from FWD-H...	**WIP**

History

#1 - 06/19/2023 09:11 AM - Alexandru Lungu

From #7363-19:

```
I see a potential optimization of CurrentTimestamp.get. This retrieves a ValueTimestampTimeZone based on Syste
m.currentTimeMillis(). However, this means that the parameters have a really high variance: the chance to have
 a cache hit is relative to the number of calls per ms. I wonder how many instances of ValueTimestampTimeZone
are created per ms and if it makes sense to cache them. From my POV:

    if there are few (< 2 per ms), we can avoid caching of course
    if there are too many (>100 per ms), we can avoid caching as this will actually pollute our cache. With an
 ideal hashing technique, the cache will be full of ValueTimestampTimeZone in 10s.

I suggest making a test on this. Please compute:

    The number of times CurrentTimestamp.get is called (in total and in avg. per ms)
    The number of ValueTimestampTimeZone in the cache once every 100 ms (a ratio considering the total size of
 the cache)
    The total time and avg. per call spent in CurrentTimestamp.get (with vs without caching)
```

As an extension to this issue, we should consider if we really need CurrentTimestamp.get that often. Maybe we can cut-off some of its usages if not absolutely necessary.

#2 - 06/19/2023 09:11 AM - Alexandru Lungu

- Related to Feature #7363: Improve H2 Value caching, hashing and equals added

#3 - 06/20/2023 06:11 AM - Alexandru Lungu

- Related to Bug #7452: Analyze memory of the value cache from FWD-H2 and eventually increase size added

#4 - 07/04/2023 07:48 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Ștefan Roman

Stefan, this is a performance concern; please focus on fixing this.

#5 - 07/05/2023 07:55 AM - Ștefan Roman

- File timestamp.patch added

I added a test for the number of CurrentTimestamp.get and the time spent doing it. I will attach a patch so you can test it with other applications.

#6 - 07/05/2023 09:59 AM - Ștefan Roman

- File cachedTimestamps.patch added

I made another test that calculates the avg ValueTimestampTimeZone that are in cache. I`m checking every 1000 Value.cache() calls, and since a lot of times the number of cached ValueTimestampTimeZone is the same, I`m only computing the average when the number changes.

#7 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to Feature #7404: Trasform replace-mode into append-mode when target table is empty added

#8 - 07/06/2023 08:40 AM - Alexandru Lungu

- Related to deleted (Feature #7404: Trasform replace-mode into append-mode when target table is empty)

#9 - 08/21/2023 05:24 AM - Alexandru Lungu

- Assignee changed from Ștefan Roman to Alexandru Lungu

#10 - 01/05/2024 04:08 AM - Alexandru Lungu

- Assignee changed from Alexandru Lungu to Alexandru Donica

#11 - 01/12/2024 08:13 AM - Alexandru Donica

I changed the CurrentTimestamp.get method to keep track of the metrics you mentioned, and roamed around the menus of a large app (app_m), to see how many times the method gets called and where from.
In a few minutes (5) of keeping the app open and opening some menus and importing some data, it would get called ~1800 times, all of the times coming from the Prepared.setCurrentRowNumber(long) -> Prepared.checkCanceled() -> Session.checkCanceled() -> Session.throttle(). The method Prepared.setCurrentRowNumber(long) is called from quite a few different places in the package h2.command.dml. During this test i did, got called from the Update class (~60 times), Select class (~1100 times), Insert class (500 times), and Delete class (~50 times).

Regarding some of the metrics, the cache was at most 5% full with ValueTimestampTimeZone values, but mostly 2%,3%, 4%, or less than 1% when not using the app much. Average time per call was for call 1750 ~0.008 ms, however it was bigger in the first few hundreds, and kept going down. The number of calls per ms is hard to calculate in a small test as the app takes a little while to start and I'm not sure if I should count that time, but since the start of the app there have been ~0.007 calls per ms (at call number 1750, but biggest recorded was at call 250 with average 0.02 calls per ms). There may be actions in the app that solicitate this method more, but I do not know which they are. I have tried opening many menus to make many select statements, and import some data from a csv, I did not try editing or deleting.
So obviously the number of calls is < 2 per ms. Nor does this method occupy much of the cache. I have not yet tried to measure average time for the execution of the method without cache, but with cache, it is ~ 0.008 ms.

#12 - 01/12/2024 08:39 AM - Alexandru Lungu

Alexandru, please keep the same profiling infrastructure you have and test with 7156b and the large POC. Please make a baseline first (how fast is the POC working without any changes). Consider running it for 100 iterations (3 or 5 times) and make an average of the total time. After, with the profiling changes, analyze how much time is spent in CurrentTimestamp.get in relation with the baseline.

For example, if the baseline works in 14seconds and the profiling shows that CurrentTimestamp.get takes ~100ms, it means that 0.7% of the time is spent there. 0.7% is quite a large number for our recent optimization efforts. Even 0.2% is worth taking in if there are slim and risk-free changes.

#13 - 01/15/2024 06:42 AM - Alexandru Donica

The average of 3 baseline runs for me was 14.665 seconds (so a total of 1_466_500 milliseconds for 100 iterations), and in 100 iterations, the method CurrentTimestamp.get was called 172_125 times, amounting to a total of 246.15 miliseconds for that method alone. Meaning this method uses ~0.0168% of the time (246.15 * 100 / 1_466_500), which seems like it's a good value, optimizing it would not improve time that much.
At most, the cache was filled 9% with ValueTimestampTimeZone values from what I noticed, but would quickly go down.
Also, there would be almost 0.1 calls/millisecond during these tests, which is < 2 per ms as you mentioned above, in the first comment. So is it worth avoiding caching after all?
Should I try to avoid caching completely for this method and try again to measure the time usage?

#14 - 01/15/2024 08:50 AM - Alexandru Lungu

I was thinking of more than just avoiding the cache and maybe remote it entirely :) However, the timing is not that concerning. 0.01% is low, especially as there are testing errors, machine delays, profiling delays, etc. 172.125 calls in 100 iterations means only 1.7212 calls per iteration (in ~14.6s). This is a very low value, especially for a method that simply does some cache look-up + update work. You can do a test without caching to check if we can get a "free meal" here, but unless this attempt buys us that ~0.01% straight-away, we can drop the effort here. The alternative is to be more invasive in FWD-H2 to gather another 0.006% improvement, which is too low.

Note: please disable cache only for ValueTimestampTimeZone coming from CurrentTimestamp.get, not the ones generated by the H2 consumer (add an avoidCache flag; default it to true, set it to false in CurrentTimestamp.get).

#15 - 01/16/2024 05:59 AM - Alexandru Donica

The improvement from removing the caching for this method is very small, 5 milliseconds faster, so a total of ~241 milliseconds for all iterations. The amount of values of type ValueTimestampTimeZone in cache has also reduced by aprox 5%.
Also, I noticed that System.currentTimeMillis() is only ~6.5% of the total time of the CurrentTimestamp.get method, so the majority of the time is spent creating the ValueTimestampTimeZone object.

#16 - 01/23/2024 03:59 AM - Alexandru Lungu

According to your findings, I think we can reject this task. The improvement is too small to tackle this. T

If Alexandru [ad] agrees, we can reject this task.

#17 - 01/23/2024 08:32 AM - Alexandru Donica

I am ok with rejecting this task.

#18 - 01/23/2024 12:22 PM - Greg Shah

- Status changed from WIP to Rejected

- % Done changed from 0 to 100

Files

timestamp.patch	1.64 KB	07/05/2023	Ștefan Roman
cachedTimestamps.patch	2.78 KB	07/05/2023	Ștefan Roman