Starburst

Inherent race in Cache invalidation

Piotr Findeisen
14.12.2023

Agenda

- Trino and caching
- What's up?
- Salvation

$(whoami)

- involved in Trino (Presto) since Jan 2017
  - "Teradata Center for Hadoop" team
  - then Starburst
- 4.5k commits (less than two a day)
  - best proof that #commits means nothing
- maintainer (comitter) since 2017
- "reviewer of the year", every year (per https://nineinchnick.github.io/trino-cicd/reports/pr/)
  - best proof that #reviews means nothing

Trino and caching

Caching history

To cache or not to cache, that's the question:
Whether 'tis wiser for code to endure
The slings of latency, or take action
Against a sea of requests and assure
To hit! Perchance to miss—ay, there's the rub.
For in that cache-hit, what bugs may come!

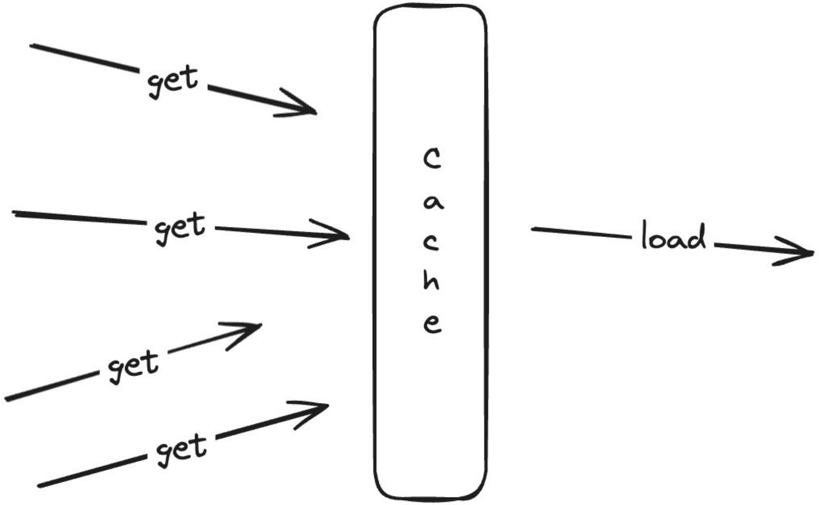
— William S., Chief Architect, The Globe

Trino caching history

- first Cache added in Trino — Oct 2012
  - Guava Cache in Raptor
- first Cache with some invalidation — Feb 2013
  - CachingHiveMetastore
  - read-only use-case (no need for invalidation)
- first pointed invalidation — Jun 2014
  - CachingHiveMetastore + Trino views support
  - read-write use-case, with invalidation
- today there are 109 Cache instances across Trino codebase
  - Guava Caches

Trino caching use-cases

- reducing remote system load
  - avoid repeated calls to a remote system
- improving latency by taking remote system calls off the critical path
  - cache with refresh interval
- reducing CPU and JVM fatigue
  - expression compilers, class generation
- ensuring read consistency
  - per-query scoped
- "load exactly once" semantics
  - e.g. task management on workers



How caches are used

```
Cache<Key, Value> cache = CacheBuilder.newBuilder()
   .maximumSize(10_000)
   // .weigher((k, v) -> …)
   .expireAfterWrite(1, TimeUnit.HOURS)
   .build();

Value value = cache.getIfPresent(key);
if (value == null) {
    value = loadValue(key);
    cache.put(key, value);
}
return value;
```

How caches are used with writes

```
// get(key):
Value value = cache
    .getIfPresent(key);
if (value == null) {
  value = loadValue(key);
  cache.put(key, value);
}
return value;
```

```
// put(key, value):
storeValue(key, value);
// or invalidateAll();
cache.invalidate(key);
```

Wisdom

> *There are only two hard things in Computer Science:*
> *cache invalidation and naming things.*
>
> — Phil Karlton, Netscape Engineer

How caches are used with writes

```
  // get(key):
  Value value = cache
1     .getIfPresent(key);
  if (value == null) {
2   value = loadValue(key);
5   cache.put(key, value);
  }
  return value;
```

```
  // put(key, value):
3 storeValue(key, value);
  // or invalidateAll();
4 cache.invalidate(key);


  // any subsequent read
  // returns old value!
6 cache.getIfPresent(key);
```

We can do better!

```
// get(key):
Value value = cache.get(
  key,
  () -> loadValue(key));
return value;
```

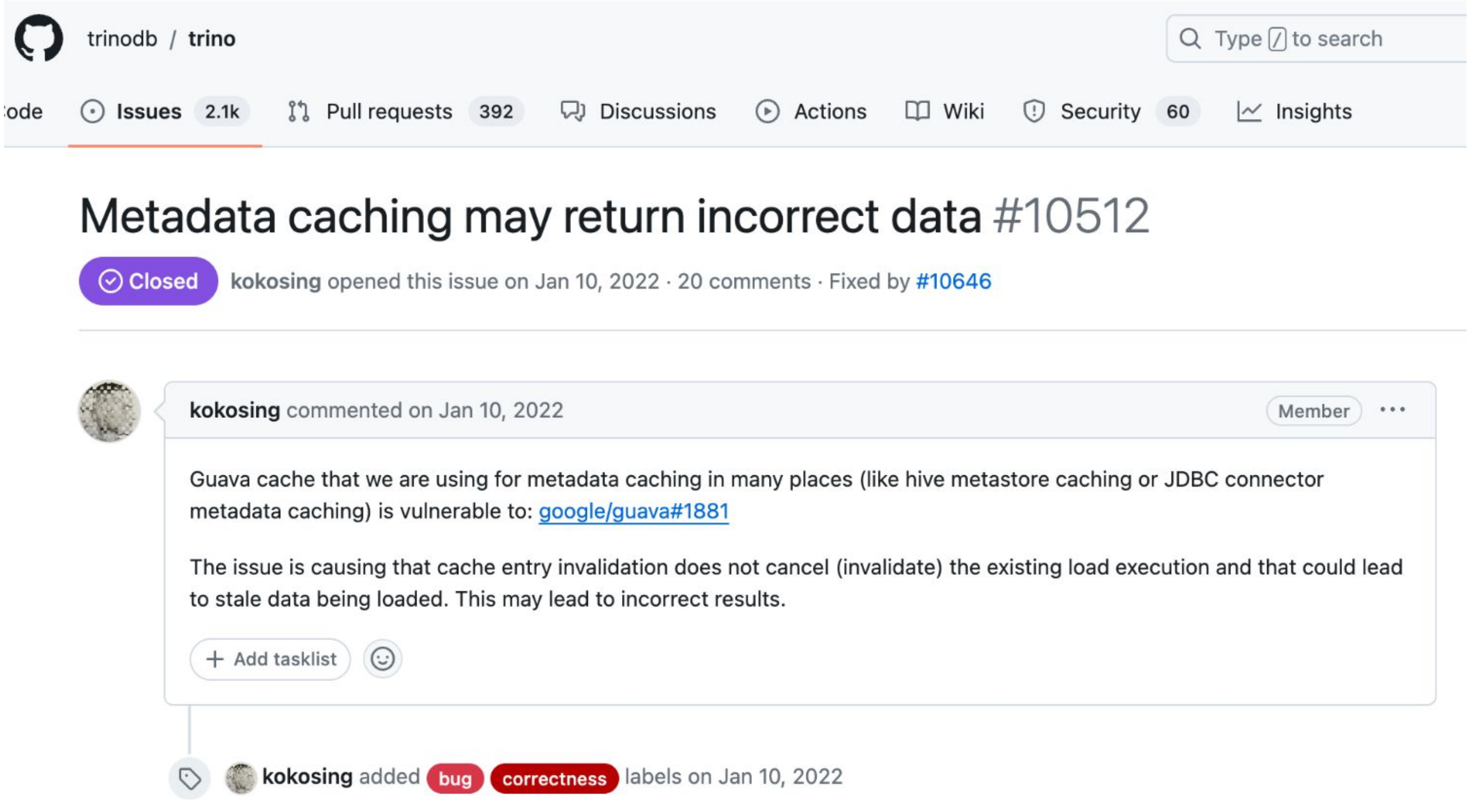
```
// put(key, value):
storeValue(key, value);
// or invalidateAll();
cache.invalidate(key);
```

Another wisdom

> *There are only two hard things in Computer Science: cache invalidation and naming things.*
>
> — Phil Karlton, Netscape Engineer

Trino Zonk :(

#10512



Metadata caching may return incorrect data #10512

Closed kokosing opened this issue on Jan 10, 2022 · 20 comments · Fixed by #10646

kokosing commented on Jan 10, 2022 Member

Guava cache that we are using for metadata caching in many places (like hive metastore caching or JDBC connector metadata caching) is vulnerable to: google/guava#1881

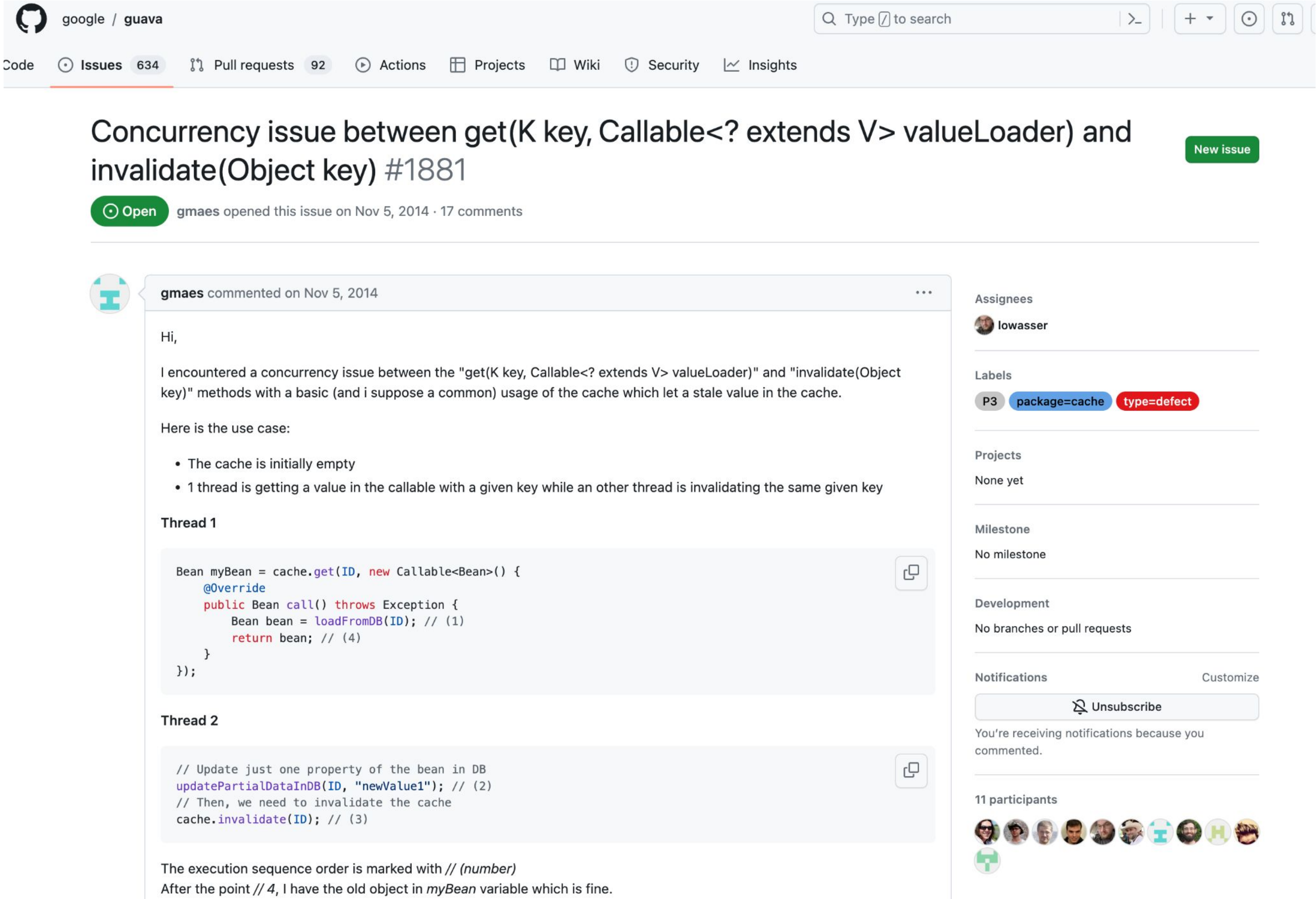
The issue is causing that cache entry invalidation does not cancel (invalidate) the existing load execution and that could lead to stale data being loaded. This may lead to incorrect results.

kokosing added bug correctness labels on Jan 10, 2022

Starburst

Guava Zonk :(

#1881



Concurrency issue between get(K key, Callable<? extends V> valueLoader) and invalidate(Object key) #1881

Open gmaes opened this issue on Nov 5, 2014 · 17 comments

gmaes commented on Nov 5, 2014

Hi,

I encountered a concurrency issue between the "get(K key, Callable<? extends V> valueLoader)" and "invalidate(Object key)" methods with a basic (and i suppose a common) usage of the cache which let a stale value in the cache.

Here is the use case:

- The cache is initially empty
- 1 thread is getting a value in the callable with a given key while an other thread is invalidating the same given key

Thread 1

```
Bean myBean = cache.get(ID, new Callable<Bean>() {
    @Override
    public Bean call() throws Exception {
        Bean bean = loadFromDB(ID); // (1)
        return bean; // (4)
    }
});
```

Thread 2

```
// Update just one property of the bean in DB
updatePartialDataInDB(ID, "newValue1"); // (2)
// Then, we need to invalidate the cache
cache.invalidate(ID); // (3)
```

The execution sequence order is marked with *// (number)*
After the point *// 4*, I have the old object in *myBean* variable which is fine.

Assignees
lowasser

Labels
P3 package=cache type=defect

Projects
None yet

Milestone
No milestone

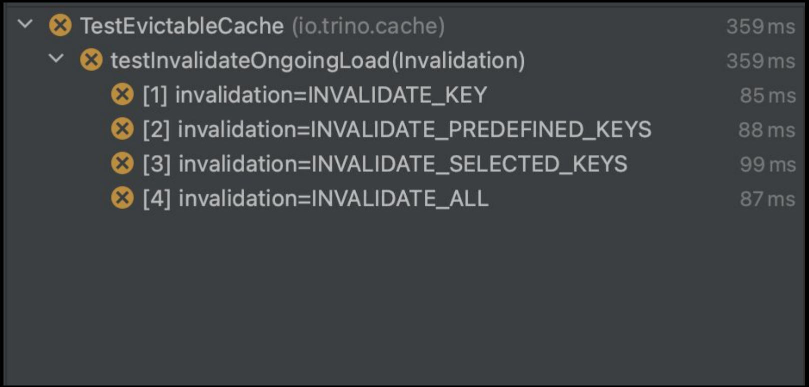
Development
No branches or pull requests

Notifications Customize
Unsubscribe
You're receiving notifications because you commented.

11 participants

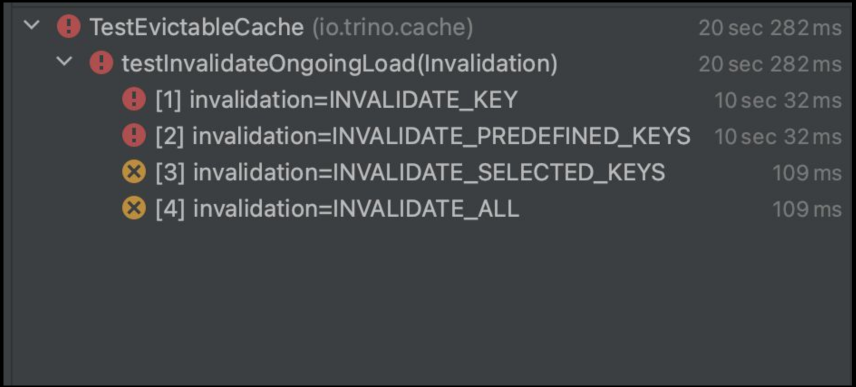
Solution: use a library!

Guava



```
⌄ ⊗ TestEvictableCache (io.trino.cache)                       359 ms
  ⌄ ⊗ testInvalidateOngoingLoad(Invalidation)                 359 ms
      ⊗ [1] invalidation=INVALIDATE_KEY                        85 ms
      ⊗ [2] invalidation=INVALIDATE_PREDEFINED_KEYS            88 ms
      ⊗ [3] invalidation=INVALIDATE_SELECTED_KEYS              99 ms
      ⊗ [4] invalidation=INVALIDATE_ALL                        87 ms
```

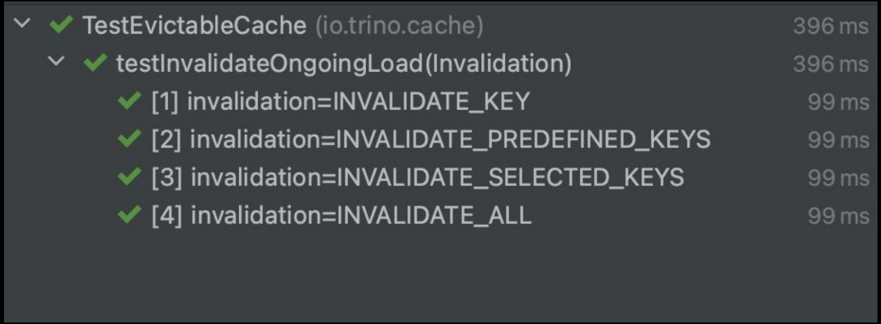
Caffeine



```
⌄ ❗ TestEvictableCache (io.trino.cache)                       20 sec 282 ms
  ⌄ ❗ testInvalidateOngoingLoad(Invalidation)                 20 sec 282 ms
      ❗ [1] invalidation=INVALIDATE_KEY                        10 sec 32 ms
      ❗ [2] invalidation=INVALIDATE_PREDEFINED_KEYS            10 sec 32 ms
      ⊗ [3] invalidation=INVALIDATE_SELECTED_KEYS                  109 ms
      ⊗ [4] invalidation=INVALIDATE_ALL                            109 ms
```

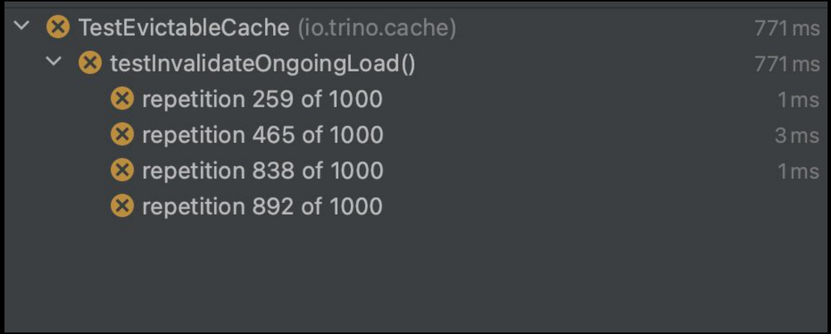
Solution: use a library!

Caffeine AsyncCache



```
˅ ✓ TestEvictableCache (io.trino.cache)                          396 ms
  ˅ ✓ testInvalidateOngoingLoad(Invalidation)                    396 ms
      ✓ [1] invalidation=INVALIDATE_KEY                           99 ms
      ✓ [2] invalidation=INVALIDATE_PREDEFINED_KEYS               99 ms
      ✓ [3] invalidation=INVALIDATE_SELECTED_KEYS                 99 ms
      ✓ [4] invalidation=INVALIDATE_ALL                           99 ms
```

… devil is in the details



```
˅ ✗ TestEvictableCache (io.trino.cache)                          771 ms
  ˅ ✗ testInvalidateOngoingLoad()                                771 ms
      ✗ repetition 259 of 1000                                     1 ms
      ✗ repetition 465 of 1000                                     3 ms
      ✗ repetition 838 of 1000                                     1 ms
      ✗ repetition 892 of 1000
```

Starburst

Solution: back to the Future

```
// get(key):
Future<Value> future = cache.get(key,
  () -> SettableFuture::new);
if (!future.isDone()) {
  future.set(loadValue(key));
}

return future.get();
```

- verbose
- load sharing?
- failed load leaves garbage
- refreshAfterWrite()
- weighted caches

Solution: generational tokens

```
// get(key):
Token token = tokens.get(
   key,
   () -> new Token(key));

Value value = cache.get(
   token,
   () -> loadValue(key));

return value;
```

```
// put(key, value):
storeValue(key, value);
// or invalidateAll();
tokens.invalidate(key);
```

Solution: generational tokens

```
// get(key):
Token token = tokens.get(
   key,
   () -> new Token(key));

Value value = cache.get(
   token,
   () -> loadValue(key));

return value;
```

```
cache = CacheBuilder
   .newBuilder()
   …
   .build();

tokens = new Concurrent-
   HashMap();

// and cache → tokens
// eviction propagation
// using a listener
```

Is it better than Futures???

Solution: create a library!

```
Cache<Key, Value> cache =          CacheBuilder.newBuilder()
  .maximumSize(10_000)
  // .weigher((k, v) -> …)
  .expireAfterWrite(1, TimeUnit.HOURS)
  .build();

// get(key):
Value value = cache.get(
  key,
  () -> loadValue(key));
return value;
```

Solution: create a library!

```
Cache<Key, Value> cache = EvictableCacheBuilder.newBuilder()
    .maximumSize(10_000)
    // .weigher((k, v) -> …)
    .expireAfterWrite(1, TimeUnit.HOURS)
    .build();

// get(key):
Value value = cache.get(
    key,
    () -> loadValue(key));
return value;
```

You can use it too

just grab it

```
<dependency>
   <groupId>io.trino</groupId>
   <artifactId>trino-cache</artifactId>
   <version>434</version>
</dependency>
```

… but Trino is not a library

```
<dependency>
   <groupId>io.github.findepi</groupId>
   <artifactId>evictable-cache</artifactId>
   <version>1</version>
</dependency>
```

You can use it too

User modernizer-maven-plugin for enforcement

```
<!-- see https://github.com/trinodb/trino/blob/⏎
  3d7121859c6a0a530813e2168fafb851c199506c/.mvn/⏎
  modernizer/violations.xml#L109-L123 -->

<violation>
  <name>com/google/common/cache/CacheBuilder.build:()Lcom/google/common/cache/Cache;</name>
  ...
</violation>

<violation>
  <name>com/google/common/cache/CacheBuilder.build:⏎
       (Lcom/google/common/cache/CacheLoader;)Lcom/google/common/cache/LoadingCache;</name>
  ...
</violation>
```

Q & A

Starburst