KIP-NEXT: Get rid of unnecessary read lock

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state:Under Discussion

Discussion thread: *here*

JIRA: *SensorAccess.getOrCreate should be more efficient*

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

.Improve Kafka performance in request handling and make the source code cleaner.

Public Interfaces

There's no public interfaces to be defined. The change proposed here is to remove a read lock which is completely unnecessary. The read lock is called for each request (at least) from producer, which is really expensive.

Proposed Changes

The getOrCreate of SensorAccess class under this folder: core\src\main\scala\kafka\server is using a read like this:

lock.readLock().lock()

The lock is used to prevent race condition: when a new sensor is created by a writer thread, and the "sensor" variable is not null more, but the initialization of the new sensor object is not finished, if at this moment another thread may gets the sensor object and use it before the sensor initialization is finished.

But this read lock can be avoided by firstly assigning the new sensor to a temporary variable say "tmp", and then initialized tmp, finally assign temp to the "sensor" variable. By doing this, we can get rid of the read lock.

Below code and screen shows the code change:

codechange.zip

```
27        *
28        * The later arguments are passed as methods as they are only called when the sensor is instantiated.
29        */
30      class SensorAccess {
31
32        def getOrCreate(sensorName: String, expirationTime: Long, lock: ReentrantReadWriteLock, metrics: Metrics, metricName: () => MetricName, config: () => MetricConfig, measure: () => MeasurableStat): Sensor = {
33          var sensor: Sensor = null
34
35          /* Acquire the read lock to fetch the sensor. It is safe to call getSensor from multiple threads.
36           * The read lock allows a thread to create a sensor in isolation. The thread creating the sensor
37           * will acquire the write lock and prevent the sensors from being read while they are being created.
38           * It should be sufficient to simply check if the sensor is null without acquiring a read lock but the
39           * sensor being present doesn't mean that it is fully initialized i.e. all the Metrics may not have been added.
40           * This read lock waits until the writer thread has released its lock i.e. fully initialized the sensor
41           * at which point it is safe to read
42           */
43          lock.readLock().lock()
44          try {
45            sensor = metrics.getSensor(sensorName)
46          }
47          finally {
48            lock.readLock().unlock()
49          }
50          var sensor: Sensor = metrics.getSensor(sensorName)
51
52          /* If the sensor is null, try to create it else return the existing sensor
53           * The sensor can be null, hence the null checks
54           */
55          if (sensor == null) {
56            /* Acquire a write lock because the sensor may not have been created and we only want one thread to create it.
57             * Note that multiple threads may acquire the write lock if they all see a null sensor initially
58             * In this case, the writer checks the sensor after acquiring the lock again.
59             * This is safe from Double Checked Locking because the references are read
60             * after acquiring read locks and hence they cannot see a partially published reference
61             */
62            lock.writeLock().lock()
63            try {
64              // Set the var for both sensors in case another thread has won the race to acquire the write lock. This will
65              // ensure that we initialise `ClientSensors` with non-null parameters.
66              sensor = metrics.getSensor(sensorName)
67              if (sensor == null) {
68                sensor = metrics.sensor(sensorName, config(), expirationTime)
69                var tempSensor: Sensor = metrics.sensor(sensorName, config(), expirationTime)
70                sensor.add(metricName(), measure())
71                tempSensor.add(metricName(), measure())
72                sensor = tempSensor
73              }
74            } finally {
75              lock.writeLock().unlock()
76            }
77          }
78          sensor
79        }
80      }
```

Compatibility, Deprecation, and Migration Plan

N/A

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.