package com.twitter.search.earlybird.partition;

import java.util.EnumMap;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.atomic.AtomicLong;

import com.twitter.search.common.metrics.SearchCounter;

import com.twitter.search.common.metrics.SearchLongGauge;

import com.twitter.search.common.metrics.SearchRateCounter;

import com.twitter.search.common.metrics.SearchStatsReceiver;

import com.twitter.search.common.metrics.SearchTimerStats;

import com.twitter.search.common.schema.thriftjava.ThriftIndexingEventType;

import com.twitter.search.earlybird.util.ScheduledExecutorManager;

/\*\*

\* Collection of common metrics used in the indexing, and related code.

\* We create a set/holder for them as we want to create all counters only one time, and these

\* counters can be used by both SimpleUpdateIndexer, PartitionIndexer, EarlybirdSegment, and others.

\*/

public class SearchIndexingMetricSet {

/\*\*

\* A proxy for the creation time of the "freshest" tweet that we have in the index.

\* It is used in computing the index freshness stat "earlybird\_index\_freshness\_millis".

\* - In the realtme clusters, this should match the creation time of highestStatusId.

\* - In the archive clusters, this should match the timestamp of the latest indexed day.

\*/

public final SearchLongGauge freshestTweetTimeMillis;

/\*\* The highest indexed tweet ID. Used to compute index freshness. \*/

public final SearchLongGauge highestStatusId;

/\*\*

\* The current timeslice's ID. We can compare this to indexer's exported current timeslice ID to

\* identify stuck timeslice rolls.

\*/

public final SearchLongGauge currentTimesliceId;

/\*\* The number of archive timeslices that we failed to process. \*/

public final SearchCounter archiveTimeSliceBuildFailedCounter;

/\*\* The number of times we checked a segment's size on disk. \*/

public final SearchCounter segmentSizeCheckCount;

/\*\* The number of segments that have reached their max size. \*/

public final SearchCounter maxSegmentSizeReachedCounter;

/\*\* The number of indexed tweets and the aggregate indexing latencies in microseconds. \*/

public final SearchTimerStats statusStats;

/\*\* The number of applied updates and the aggregate indexing latencies in microseconds. \*/

public final SearchTimerStats updateStats;

/\*\* The number of retried updates and the aggregate indexing latencies in microseconds. \*/

public final SearchTimerStats updateRetryStats;

/\*\* The number of applied user updates and the aggregate indexing latencies in microseconds. \*/

public final SearchTimerStats userUpdateIndexingStats;

/\*\* The number of applied userGeoScrubEvents and the aggregate indexing latencies in

\* microseconds. \*/

public final SearchTimerStats userScrubGeoIndexingStats;

/\*\* The number of updates attempted on missing tweets. \*/

public final SearchRateCounter updateOnMissingTweetCounter;

/\*\* The number of updates dropped. \*/

public final SearchRateCounter droppedUpdateEvent;

/\*\* The latencies in microseconds of the PartitionIndexer loop. \*/

public final SearchTimerStats partitionIndexerRunLoopCounter;

/\*\* The latencies in microseconds of the PartitionIndexer.indexFromReaders() calls. \*/

public final SearchTimerStats partitionIndexerIndexFromReadersCounter;

/\*\* The number of invocations of the PartitionIndexer task. \*/

public final SearchCounter partitionIndexerIterationCounter;

/\*\* The number of unsorted updates handled by SimpleUpdateIndexer. \*/

public final SearchCounter simpleUpdateIndexerUnsortedUpdateCounter;

/\*\* The number of unsorted updates with the wrong segment handled by SimpleUpdateIndexer. \*/

public final SearchCounter simpleUpdateIndexerUnsortedUpdateWithWrongSegmentCounter;

/\*\* The number of invocations of the SimpleUserUpdateIndexer task. \*/

public final SearchCounter simpleUserUpdateIndexerIterationCounter;

/\*\* The number of exceptions encountered by SimpleSegmentIndexer while indexing a segment. \*/

public final SearchCounter simpleSegmentIndexerExceptionCounter;

/\*\*

\* A map from TIE update type to the creation time of the updated tweet in milliseconds of the

\* freshest update we have indexed.

\*/

public final EnumMap<ThriftIndexingEventType, AtomicLong> updateFreshness =

new EnumMap<>(ThriftIndexingEventType.class);

public final SearchStatsReceiver searchStatsReceiver;

public static class StartupMetric {

// Switched from 0 to 1 during the event.

private SearchLongGauge duringGauge;

// Switched from 0 to time it takes, in milliseconds.

private SearchLongGauge durationMillisGauge;

StartupMetric(String name) {

this.duringGauge = SearchLongGauge.export(name);

this.durationMillisGauge = SearchLongGauge.export("duration\_of\_" + name);

}

public void begin() {

duringGauge.set(1);

}

public void end(long durationInMillis) {

duringGauge.set(0);

durationMillisGauge.set(durationInMillis);

}

}

public final StartupMetric startupInProgress;

public final StartupMetric startupInIndexCompletedSegments;

public final StartupMetric startupInLoadCompletedSegments;

public final StartupMetric startupInIndexUpdatesForCompletedSegments;

public final StartupMetric startupInCurrentSegment;

public final StartupMetric startupInUserUpdates;

public final StartupMetric startupInQueryCacheUpdates;

public final StartupMetric startupInMultiSegmentTermDictionaryUpdates;

public final StartupMetric startupInWarmUp;

// Kafka metrics

public final StartupMetric startupInLoadFlushedIndex;

public final StartupMetric startupInFreshStartup;

public final StartupMetric startupInIngestUntilCurrent;

public final StartupMetric startupInUserUpdatesStartup;

public final StartupMetric startupInUserEventIndexer;

public final StartupMetric startupInAudioSpaceEventIndexer;

public SearchIndexingMetricSet(SearchStatsReceiver searchStatsReceiver) {

this.freshestTweetTimeMillis = searchStatsReceiver.getLongGauge(

"earlybird\_freshest\_tweet\_timestamp\_millis");

this.highestStatusId = searchStatsReceiver.getLongGauge("highest\_indexed\_status\_id");

this.currentTimesliceId = searchStatsReceiver.getLongGauge("earlybird\_current\_timeslice\_id");

this.archiveTimeSliceBuildFailedCounter = searchStatsReceiver.getCounter(

"archive\_time\_slice\_build\_failed");

this.segmentSizeCheckCount = searchStatsReceiver.getCounter("segment\_size\_check\_count");

this.maxSegmentSizeReachedCounter = searchStatsReceiver.getCounter("max\_segment\_reached");

this.statusStats = searchStatsReceiver.getTimerStats(

"index\_status", TimeUnit.MICROSECONDS, false, false, false);

this.updateStats = searchStatsReceiver.getTimerStats(

"updates", TimeUnit.MICROSECONDS, false, false, false);

this.updateRetryStats = searchStatsReceiver.getTimerStats(

"update\_retries", TimeUnit.MICROSECONDS, false, false, false);

this.userUpdateIndexingStats = searchStatsReceiver.getTimerStats(

"user\_updates", TimeUnit.MICROSECONDS, false, false, false);

this.userScrubGeoIndexingStats = searchStatsReceiver.getTimerStats(

"user\_scrub\_geo", TimeUnit.MICROSECONDS, false, false, false);

this.updateOnMissingTweetCounter = searchStatsReceiver.getRateCounter(

"index\_update\_on\_missing\_tweet");

this.droppedUpdateEvent = searchStatsReceiver.getRateCounter("dropped\_update\_event");

this.partitionIndexerRunLoopCounter = searchStatsReceiver.getTimerStats(

"partition\_indexer\_run\_loop", TimeUnit.MICROSECONDS, false, true, false);

this.partitionIndexerIndexFromReadersCounter = searchStatsReceiver.getTimerStats(

"partition\_indexer\_indexFromReaders", TimeUnit.MICROSECONDS, false, true, false);

this.partitionIndexerIterationCounter = searchStatsReceiver.getCounter(

ScheduledExecutorManager.SCHEDULED\_EXECUTOR\_TASK\_PREFIX + "PartitionIndexer");

this.simpleUpdateIndexerUnsortedUpdateCounter = searchStatsReceiver.getCounter(

"simple\_update\_indexer\_unsorted\_update\_count");

this.simpleUpdateIndexerUnsortedUpdateWithWrongSegmentCounter = searchStatsReceiver.getCounter(

"simple\_update\_indexer\_unsorted\_update\_with\_wrong\_segment\_count");

this.simpleUserUpdateIndexerIterationCounter = searchStatsReceiver.getCounter(

ScheduledExecutorManager.SCHEDULED\_EXECUTOR\_TASK\_PREFIX + "SimpleUserUpdateIndexer");

this.simpleSegmentIndexerExceptionCounter = searchStatsReceiver.getCounter(

"exception\_while\_indexing\_segment");

for (ThriftIndexingEventType type : ThriftIndexingEventType.values()) {

AtomicLong freshness = new AtomicLong(0);

updateFreshness.put(type, freshness);

String statName = ("index\_freshness\_" + type + "\_age\_millis").toLowerCase();

searchStatsReceiver.getCustomGauge(statName,

() -> System.currentTimeMillis() - freshness.get());

}

this.startupInProgress = new StartupMetric("startup\_in\_progress");

this.startupInIndexCompletedSegments = new StartupMetric("startup\_in\_index\_completed\_segments");

this.startupInLoadCompletedSegments = new StartupMetric("startup\_in\_load\_completed\_segments");

this.startupInIndexUpdatesForCompletedSegments =

new StartupMetric("startup\_in\_index\_updates\_for\_completed\_segments");

this.startupInCurrentSegment = new StartupMetric("startup\_in\_current\_segment");

this.startupInUserUpdates = new StartupMetric("startup\_in\_user\_updates");

this.startupInQueryCacheUpdates = new StartupMetric("startup\_in\_query\_cache\_updates");

this.startupInMultiSegmentTermDictionaryUpdates =

new StartupMetric("startup\_in\_multi\_segment\_dictionary\_updates");

this.startupInWarmUp = new StartupMetric("startup\_in\_warm\_up");

this.startupInLoadFlushedIndex = new StartupMetric("startup\_in\_load\_flushed\_index");

this.startupInFreshStartup = new StartupMetric("startup\_in\_fresh\_startup");

this.startupInIngestUntilCurrent = new StartupMetric("startup\_in\_ingest\_until\_current");

this.startupInUserUpdatesStartup = new StartupMetric("startup\_in\_user\_updates\_startup");

this.startupInUserEventIndexer = new StartupMetric("startup\_in\_user\_events\_indexer");

this.startupInAudioSpaceEventIndexer =

new StartupMetric("startup\_in\_audio\_space\_events\_indexer");

searchStatsReceiver.getCustomGauge("earlybird\_index\_freshness\_millis",

this::getIndexFreshnessInMillis);

this.searchStatsReceiver = searchStatsReceiver;

}

long getIndexFreshnessInMillis() {

return System.currentTimeMillis() - freshestTweetTimeMillis.get();

}

}