KIP-869: Improve Streams State Restoration Visibility

Status

Current state: Accepted

JIRA:

Unable to render Jira issues macro, execution error.

Unable to render Jira issues macro, execution error.

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

Motivation

State restoration is a key procedure in Kafka Streams when processing tasks are migrated in a rebalance, as well for maintaining standby tasks for failure recoveries.

This proposal aims to expose more visibilities around this procedure to users, and is composed of two components: 1) augmenting the metrics related to restoration, 2) add new APIs for **StateRestoreListener.**

Public Interfaces

Below summarizes the public API changes in this KIP.

Restoration metrics

We propose add metrics both on the thread-level (default reporting level is INFO) as well as on the task level (default reporting level is DEBUG).

Note that we will have separate thread handling restoration procedures, and hence their thread id would be different from stream threads.

Thread-level metric tags are:

- type=stream-state-updater-metrics
- thread-id=[threadId]

Task-level metric tags are:

- type=stream-task-metrics
- thread-id=[threadId]
- task-id=[taskId]

The POC implementation of the proposed metrics can be found here: https://github.com/apache/kafka/pull/12391

Metric Name	Level	Type	Description	Notes

active-restoring-tasks	thread / INFO	count	The number of active tasks currently undergoing restoration	
standby-updating-tasks	thread / INFO	count	The number of active tasks currently undergoing updating	
active-paused-tasks	thread / INFO	count	The number of active tasks paused restoring	
standby-paused-tasks	thread / INFO	count	The number of standby tasks paused updating	
idle-ratio	thread / INFO	gauge (percentage)	The fraction of time the thread spent on being idle	idle-ratio + restore/update-ratio + checkpoint-ratio should be 1
active-restore-ratio	thread / INFO	gauge (percentage)	The fraction of time the thread spent on restoring active tasks	idle-ratio + restore/update-ratio + checkpoint-ratio should be 1; only one of the restore/update-ratio should be non-zero
standby-update-ratio	thread / INFO	gauge (percentage)	The fraction of time the thread spent on updating standby tasks	idle-ratio + restore/update-ratio + checkpoint-ratio should be 1; only one of the restore/update-ratio should be non-zero
checkpoint-ratio	thread / INFO	gauge (percentage)	The fraction of time the thread spent on checkpointing restored progress	idle-ratio + restore/update-ratio + checkpoint-ratio should be 1
restore-records-rate	thread / INFO	rate	The average per-second number of records restored/updated for all tasks	
restore-call-rate	thread / INFO	rate	The average per-second number of restore calls triggered	
restore-total	task / DEBUG	count	The total number of records processed during restoration for active task	the metric would persist even when the task completed restoration, and would be removed only when the task is removed from the thread.
restore-rate	task / DEBUG	rate	The average per-second number of records restored for active task	the metric would drop to zero when the task completed restoration, and would be removed only when the task is removed from the thread.
update-total	task / DEBUG	count	The total number of records updated for standby task	same as above
update-rate	task / DEBUG	rate	The average per-second number of records updated for standby task	same as above
restore-remaining-records-total	task / INFO	count	The number of records remained to be restored for active tasks	

Along with these new metrics, we would also deprecate the metrics below:

Metric Name	Type	Description	Notes
standby-process-ratio	gauge	Task-level; the fraction of time the processing thread spent on processing this standby task	Removed since standby tasks are not processed by stream thread

New Method in StateRestoreListener

When an active task starts restoration, **StateStoreListener#onRestoreStart** would be triggered. The restoring task could end in two possible ways:

1) Restoration completes and the task could now be processed normally with incoming stream records. At this time **StateStoreListener#onRestoreEnd** would be triggered.

2) Restoration was paused before completes, e.g. since another rebalance is triggered and this task is suspended and potentially migrated out of the current host later. At this time no callbacks would be triggered.

We propose to cover the second case above with a new API, so that each **onRestoreStart** function would be paired with either an **onRestoreEnd** function or an **onRestorePaused** function. Note that if the suspended task was re-assigned back to the current host, another **onRestoreStart** would be triggered again.

```
public interface StateRestoreListener {

    void onRestoreStart(final TopicPartition topicPartition,
                        final String storeName,
                        final long startingOffset,
                        final long endingOffset);

    void onRestoreEnd(final TopicPartition topicPartition,
                      final String storeName,
                      final long totalRestored);

    ...

    /**
     * NEW FUNC. Method called when restoring the {@link StateStore} is suspended due to the task being
suspended from the host.
     *           If the task was resumed after suspension and restoration continues, another {@link
onRestoreStart} would be called.
     */
    default void onRestoreSuspended(final TopicPartition topicPartition,
                                    final String storeName,
                                    final long totalRestored) {
        // do nothing
    }
}
```

Compatibility, Deprecation, and Migration Plan

- *The default implementation of the new onRestorePaused function would be a no-op, to maintain backward compatibilities.*
- *Deprecated metric would still be exposed, and only be removed in the next major release.*

Rejected Alternatives

None.