# KIP-607: Add Metrics to Kafka Streams to Report Properties of RocksDB



## Status

**Current state**: Accepted

**Discussion thread**: http://mail-archives.apache.org/mod_mbox/kafka-dev/202005.mbox/%3CCADR0NwzJBJa6WihnpmGj0R%2BYPVrojq4Kg_hOArNEytHAG-tZAQ%40mail.gmail.com%3E

**JIRA**: 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

Monitoring RocksDB instances run in a Kafka Streams application allows to react to increased memory and disk demands of RocksDB as well as to other performance issues related to RocksDB before the application crashes. Additionally, such monitoring are useful for analysing the cause of a crash. Currently, the metrics exposed by Kafka Streams include information about RocksDB instances run by an application (see KIP-471: Expose RocksDB Metrics in Kafka Streams for more details) but they do not provide any information about memory or disk usage of the RocksDB instances. Moreover, the metrics in KIP-471 expose statistics collected in RocksDB. Collecting statistics in RocksDB may have an impact on performance. That is also the reason why the metrics in KIP-471 are on recording level DEBUG. This KIP proposes to add metrics to Kafka Streams that report properties that RocksDB exposes by default and consequently can be exposed on recording level INFO. The metrics in this KIP and KIP-471 complement each other.

## Public Interfaces

Each added metric will be on **store-level** and have the following tags:

- type = stream-state-metrics
- thread-id = [thread ID]
- task-id = [task ID]
- rocksdb-state-id = [store ID]  for key-value stores
- rocksdb-session-state-id = [store ID]  for session stores
- rocksdb-window-state-id = [store ID]  for window stores

The following metrics will be exposed in the Kafka Streams' metrics

- num-immutable-mem-table
- cur-size-active-mem-table
- cur-size-all-mem-tables
- size-all-mem-tables
- num-entries-active-mem-table
- num-entries-imm-mem-tables
- num-deletes-active-mem-table
- num-deletes-imm-mem-tables
- mem-table-flush-pending
- num-running-flushes
- compaction-pending
- num-running-compactions
- estimate-pending-compaction-bytes
- total-sst-files-size
- live-sst-files-size
- num-live-versions
- block-cache-capacity
- block-cache-usage
- block-cache-pinned-usage
- estimate-num-keys
- estimate-table-readers-mem
- background-errors

The recording level for all metrics will be **INFO**

# Proposed Changes

In this section, we will explain the meaning of the metrics listed in the previous section. To better understand the metrics, some basic concepts of RocksDB need to be explained first.

- **Memtable:** Memtables are in-memory write buffers. Each new key-value pair is first written to a memtable and each read looks first into the memtable before it looks on disk. Once a memtable is full it becomes immutable and it is replaced by a new memtable. A background thread flushes a memtable asynchronously to disk. Additionally, memtables can also be flushed manually. RocksDB keeps in memory the currently active memtables, full but not yet flushed memtables, and flushed memtables that are kept around to maintain write history in memory.
- **Compaction:** From time to time RocksDB needs to clean up the data it stores on disk and bring is LSM tree into a good shape (see https://github.com/facebook/rocksdb/wiki/Compaction). Compactions might block writes and flushes. Additionally, RocksDB offers different compaction algorithms with different properties. Thus, it is a good practise to monitor compactions in RocksDB.
- **SST files:** SST files are the files in which RocksDB stores the data on disk. SST stands for Sorted Sequence Table.
- **Version**: A version consists of all the live SST files at one point of time. Once a flush or compaction finishes, a new version is created because the list of live SST files has changed. An old version can be used by on-going read requests or compaction jobs. Old versions will eventually be garbage collected.
- **Cache:** RocksDB caches data in memory for reads. By default, those caches contain only data blocks, i.e., uncompressed sequences of key-value pairs in sorted order. Therefore this cache is often referred to as block cache. However, users can configure RocksDB to also store index and filter blocks in the cache.

The names of the metrics are taken from the following list in the RocksDB repo (with "rocksdb." prefix ripped off):
https://github.com/facebook/rocksdb/blob/b9a4a10659969c71e6f6eab4e4bae8c36ede919f/include/rocksdb/db.h#L654-L686.

Those are public RocksDB properties. We decided to keep the RocksDB names to avoid a mapping that users need to look up or to memorize.

### num-immutable-mem-table

Number of immutable memtables that have not yet been flushed. For segmented state stores, the sum of the number of immutable memtables over all segments is reported.

### cur-size-active-mem-table

Approximate size of active memtable in bytes. For segmented state stores, the sum of the sizes over all segments is reported.

### cur-size-all-mem-tables

Approximate size of active and unflushed immutable memtable in bytes. For segmented state stores, the sum of sizes over all segments is reported.

### size-all-mem-tables

Approximate size of active, unflushed immutable, and pinned immutable memtables in bytes. Pinned immutable memtables are flushed memtables that are kept in memory to maintain write history in memory. For segmented state stores, the sum of sizes over all segments is reported.

### num-entries-active-mem-table

Total number of entries in the active memtable. For segmented state stores, the sum of number of entries over all segments is reported.

### num-entries-imm-mem-tables

Total number of entries in the unflushed immutable memtables. For segmented state stores, the sum of number of entries over all segments is reported.

### num-deletes-active-mem-table

Total number of delete entries in the active memtable. For segmented state stores, the sum of number of deletes over all segments is reported.

### num-deletes-imm-mem-tables

Total number of delete entries in the unflushed immutable memtables. For segmented state stores, the sum of number of deletes over all segments is reported.

### mem-table-flush-pending

This metric returns 1 if a memtable flush is pending; otherwhise it returns 0. For segmented state stores, the sum of pending flushes over all segments is reported.

### num-running-flushes

Number of currently running flushes. For segmented state stores, the sum of running flushes over all segments is reported.

### compaction-pending

This metric 1 if at least one compaction is pending; otherwise, the metric reports 0. For segmented state stores, the sum of ones and zeros over all segments is reported.

### num-running-compactions

Number of currently running compactions. For segmented state stores, the sum of the number of currently running compactions over all segments is reported.

### estimate-pending-compaction-bytes

Estimated total number of bytes a compaction needs to rewrite on disk to get all levels down to under target size. In other words, this metrics relates to the write amplification in level compaction. Thus, this metric is not valid for compactions other than level-based. For segmented state stores, the sum of the estimated total number of bytes over all segments is reported.

### total-sst-files-size

Total size in bytes of all SST files. For segmented state stores, the sum of the sizes of SST files over all segments is reported.

### live-sst-files-size

Total size in bytes of all SST files that belong to the latest LSM tree. For segmented state stores, the sum of the sizes of SST files over all segments is reported.

### num-live-versions

Number of live versions. More live versions often mean more SST files are held from being deleted, by iterators or unfinished compactions. For segmented state stores, the sum of the number of versions over all segments is reported.

### block-cache-capacity

Block cache capacity. For segmented state stores, the sum of the cache capacity over all segments is reported, if separate caches are used, otherwise, if only one cache is used, the cache capacity of any segment is reported.

### block-cache-usage

Memory size for the entries residing in block cache. For segmented state stores, the sum of the cache capacity over all segments is reported, if separate caches are used, otherwise, if only one cache is used, the cache capacity of any segment is reported.

### block-cache-pinned-usage

Memory size for the entries being pinned. For segmented state stores, the sum of the cache capacity over all segments is reported, if separate caches are used, otherwise, if only one cache is used, the cache capacity of any segment is reported.

### estimate-num-keys

Estimated number of total keys in the active and unflushed immutable memtables and storage. For segmented state stores, the sum of the estimated number of keys over all segments is reported.

### estimate-table-readers-mem

Estimated memory in bytes used for reading SST tables, excluding memory used in block cache (e.g., filter and index blocks). This metric records the memory used by iterators as well as filters and indices if the filters and indices are not maintained in the block cache. Basically this metric reports the memory used outside the block cache to read data. For segmented state stores, the sum of the estimated memory over all segments is reported.

### background-errors

Accumulated number of background errors. For segmented state stores, the sum of the number of background errors over all segments is reported.

## Examples

1. If users want to monitor the total memory usage of RocksDB, they should compute size-all-mem-tables + block-cache-usage + estimate-table-readers-mem. All of this memory is off-heap memory, i.e., it is not managed by the JVM. Note, that the monitored total memory usage is an estimation. Users can bound the total memory usage by configuring RocksDB as described in https://kafka.apache.org/25/documentation/streams/developer-guide/memory-mgmt.html#rocksdb
2. With mem-table-flush-pending and num-running-flushes, users can monitor the flushing behavior of their state stores. Similarly, users can monitor the compaction behavior of their state stores with compaction-pending and num-running-compactions.
3. To monitor the sizes of the LSM trees used in their state stores, users can monitor total-sst-files-size.

## Performance Consideration

All the metrics will be implemented as gauges. That means, the metrics would not be recorded if the metrics reporting system used by the user does not query the metric. Hence, the number of metrics presented in this KIP should neither affect the performance of the RocksDB instances nor the performance of Kafka Streams if they are not queried.

# Compatibility, Deprecation, and Migration Plan

Since metrics are only added and no other metrics are modified, this KIP should not

- affect backward-compatibility
- deprecate public interfaces
- need a migration plan other than adding the new metrics to its own monitoring component

# Rejected Alternatives

### Introduce configuration in Kafka Streams to name RocksDB properties to expose

Since all of the above metrics can be exposed as gauges, there should not be too much performance overhead because recording is only triggered when the metric is actually queried. We thought that the maintenance costs of a configuration would be higher than just exposing this set of RocksDB properties.