KIP-116: Add State Store Checkpoint Interval Configuration

Status

Current state: *Rejected*

Discussion thread: *here*

JIRA: *here*

Released:

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

Motivation

When a Kafka Streams application initializes a state store it checks for the existence of a checkpoint file. The checkpoint contains the offsets for the latest records that have been both flushed to the state store and written to the change-log. If the checkpoint file exists then state store restoration begins from the offsets found in the checkpoint. If the checkpoint doesn't exist then restoration starts from the earliest offset. Currently the checkpoint file is only written on a clean shutdown, and is deleted at startup. In the event of a hard failure the checkpoint is lost and the stores will be recovered from scratch on the next restart - this can be a lengthy process and lead to significant delays during application startup.

To reduce the restart costs we can regularly write the checkpoint file on a user defined interval. This will allow users some control over the time it takes to restart stateful Kafka Streams applications.

Public Interfaces

We will add the following configuration option to *StreamsConfig*:

```
public static final String STATESTORE_CHECKPOINT_INTERVAL_MS_CONFIG = "statestore.checkpoint.interval.ms";
```

This parameter will control the minimum frequency at which the checkpoint files will be written. The default value will be set to 5 minutes.

Proposed Changes

We will add the above config parameter to *StreamsConfig*. During *StreamTask#commit()*, *StandbyTask#commit()*, and *GlobalUpdateStateTask#flushState()* we will check if the checkpoint interval has elapsed and write the checkpoint file. If the checkpoint interval hasn't elapsed then nothing is written.

As this is done only on commit, the minimum checkpoint interval will be the value of *commit.interval.ms*. In effect the actual checkpoint interval will be a multiple of the commit interval. This is required as we explicitly flush the state stores on commit and we need to ensure that any offsets we checkpoint have been both flushed to disk and written to any change-logs that exist.

Compatibility, Deprecation, and Migration Plan

- Existing users may need to adjust the configuration parameter depending on load.

Test Plan

Aside from unit and integration tests we will do some load/performance testing to measure the overhead and configure a sensible default value.

Rejected Alternatives

- Use RocksDB checkpoints: Complicated to maintain and only good for RocksDB