KIP-816: Topology changes without local state reset

Status

Current state: Under Discussion

Discussion thread: here

JIRA: KAFKA-13627

Motivation

When changes are made to a Topology that modifies its structure, users must use the Application Reset tool to reset the local state of their application prior to deploying the change. Consequently, these changes require rebuilding all local state stores from their changelog topics in Kafka.

The time and cost of rebuilding state stores is determined by the size of the state stores, and their recent write history, as rebuilding a store entails replaying all recent writes to the store. For applications that have very large stores, or stores with extremely high write-rates, the time and cost of rebuilding all state in the application can be prohibitively expensive. This is a significant barrier to building highly scalable applications with good availability.

Changes to the Topology that do not directly affect a state store should not require the local state of that store to be reset/deleted. This would allow applications to scale to very large data sets, whilst permitting the application behaviour to evolve over time.

Background

Tasks in a Kafka Streams Topology are logically grouped by “Topic Group" (aka. Subtopology). Topic Groups are assigned an ordinal (number), based on their position in the Topology. This Topic Group ordinal is used as the prefix for all Task IDs: `<topic-group-ordinal>_<partition-number>`, e.g. `2_14`

If new Topic Groups are added, old Topic Groups are removed, or existing Topic Groups are re-arranged, this can cause the assignment of ordinals to change *even for Topic Groups that have not been modified*.

When the assignment of ordinals to Topic Groups changes, existing Tasks are invalidated, as they no longer correspond to the correct Topic Groups. Local state is located in directories that include the Task ID (e.g. `/state/dir/2_14/mystore/rocksdb/…`), and since the Tasks have all been invalidated, all existing local state directories are also invalid.

Attempting to start an application that has undergone these ordinal changes, without first clearing the local state, will cause Kafka Streams to attempt to use the existing local state for the wrong Tasks. Kafka Streams detects this discrepancy and prevents the application from starting.

Non-goals

It is not a goal to permit different versions of a Kafka Streams Topology to co-exist in the same Consumer Group.

We only intend to allow Kafka Streams to maximise use of existing local state when deploying a new version of a Topology. When deploying such upgrades, the entire cluster will still need to be brought down and brought back up together, instead of a rolling restart.

Proposed Changes

New Configuration Properties

Property	Default	Description
`automatic.state.relocation`	`true`	When `true`, on startup, Kafka Streams will automatically relocate state stores it finds on-disk that are no longer in the correct location. This can occur after changes to a Topology that re-order Sub-Topologies such that Task IDs for existing state are no longer valid.

Implementation Details

When `KafkaStreams#start()` is called, if `automatic.state.relocation` is enabled, Streams will automatically search the configured state directory and move Task state directories to the correct locations according to the current Topology graph.

The "correct" location is defined as: All components of the Path are the same as the current Path, except for the sub-topology ordinal, which is determined from the current Topology graph, by looking for the sub-topology that references the store, by-name.

Example movement

`state.dir/`**`2`**`_14/rocksdb/mystore state.dir/`**`3`**`_14/rocksdb/mystore`

Compatibility, Deprecation, and Migration Plan

This change is backwards compatible with previous versions of Kafka Streams. No deprecation or migration is necessary.

The `automatic.state.relocation` configuration is provided to enable users to disable this behaviour if they need to, but is enabled by default because it should not cause problems for any users.

Moving directories in all modern filesystems is a very cheap operation and O(1), so the performance impact on application start-up, even for application instances with a large number of stateful Tasks, should be minimal.

Rejected Alternatives

Several alternative solutions were explored, but were found to require considerable changes to the internals of Kafka Streams, making them difficult to implement safely.

Alternative 1: De-couple local state directories from Task ID

Local state will be stored under a revised directory structure that no longer includes Task ID in the path.

The current directory structure is: `/state/dir/<task id>/rocksdb/<store name>`

The new directory structure will be: `/state/dir/<store name>/<partition number>/rocksdb`

All local state, including `.checkpoint` files will be moved under this new path.

Existing Task ID directories will still be used for Task-specific state; notably `.lock` files.

When a Kafka Streams cluster assigns Tasks to members, any existing local state is used as a factor to determine which member is assigned each Task. The mapping of StateStore to Task is already known during assignment, and will be used in an updated assignment algorithm that uses the new StateStore paths to inform Task assignment.

Migration

A migration tool will enable users to automatically move existing local state from the old locations to the new locations. This tool must be used between shutting down the old version of the application and starting up the new version, during an upgrade of Kafka Streams.

Users that choose not to use this tool to migrate their local state will need to reset their application's state, to ensure that stale local state does not remain under old paths.

Challenges

- Much of the Kafka Streams codebase assumes that TaskId is encoded directly in the state directory path. See `StateDirectory`.
- Task-wide `.checkpoint` files, which can contain entries for *multiple* state stores, are currently stored in the task directory. If we decouple the directory from specific Tasks, we will need to either find another way to store these files, or retain the existing task directories exclusively for Task-wide meta-data.

Alternative 2: Change Task ID prefix from an ordinal to a stable hash

The instability of the Task Group ordinal is the reason that existing local state becomes invalidated when the Topology changes. Using a more stable identifier will solve this problem under *most* circumstances.

The TaskID's `topicGroupId` will be changed from an int to a 64-bit murmur2 hash of all the topics within the topic group, lexically ordered. This will be an API breaking change, as it will change the types in the API of SubscriptionInfoData.

This will ensure that local state will only be invalidated if the set of topics in an existing Subtopology is modified; and only the local state for that Subtopology will be invalidated.

When this invalidation of state *does* occur, it will not cause conflicts*, because the local state path for the new Task ID will be guaranteed* not to exist.

* There is a small risk of hash collision that will need to be addressed.

Challenges

- Still results in state going unnecessarily missing for *some* sub-topologies (i.e. when a source topic is added to an existing sub-topology).
- Hash collisions that would need to be mitigated, despite being unlikely.
- Massive changes required throughout Kafka Streams codebase.