KIP-775: Custom partitioners in foreign key joins

Status

Current state: *Accepted*

Discussion thread: *link*

JIRA: *KAFKA-13261*

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

Motivation

Today, foreign key (FK) joins in Kafka Streams only work if both tables being joined (the *primary table* and the *foreign-key table*) use the default partitioner.

This limitation is due to the fact that the subscription and response topics used in the implementation of FK joins always use the default partitioner today. If the foreign-key table is not copartitioned with the subscription topic, then FK lookups may be routed to a Streams instance that does not have state for the foreign-key table, resulting in missing join records. Similarly, if the primary table is not copartitioned with the response topic, then subscription responses may be routed to an instance that does not contain the original (triggering) record, resulting in a failed hash comparison and a dropped join result.

This KIP proposes to add support for FK joins on tables with custom partitioners, by extending the FK join interface to allow custom partitioners to be passed in.

Public Interfaces

Custom partitioners will be passed in as part of a new `TableJoined` object, analogous to the existing `Joined` and `StreamJoined` objects. `TableJoinec` will contain both partitioners and will also implement `NamedOperation`. Existing FK join methods that take `Named` as a parameter will be deprecated and replaced with new versions that accept `TableJoined` instead.

The following `KTable` interfaces will be added. They are analogous to existing methods that accept `Named`, except with `Named` replaced by `TableJoined`. The existing methods which accept `Named` will be marked for deprecation (to be removed in the next major release, likely 4.0).

```
    /**
     * Join records of this {@code KTable} with another {@code KTable} using non-windowed inner join,
     * using the {@link TableJoined} instance for configuration of the {@link StreamPartitioner this table's
     * key serde} and {@link StreamPartitioner the other table's key serde}.
     * <p>
     * This is a foreign key join, where the joining key is determined by the {@code foreignKeyExtractor}.
     *
     * @param other               the other {@code KTable} to be joined with this {@code KTable}. Keyed by KO.
     * @param foreignKeyExtractor a {@link Function} that extracts the key (KO) from this table's value (V). If
the
     *                            result is null, the update is ignored as invalid.
     * @param joiner              a {@link ValueJoiner} that computes the join result for a pair of matching
records
     * @param tableJoined         a {@link TableJoined} used to configure partitioners and names of internal
topics and stores
     * @param <VR>                the value type of the result {@code KTable}
     * @param <KO>                the key type of the other {@code KTable}
     * @param <VO>                the value type of the other {@code KTable}
     * @return a {@code KTable} that contains the result of joining this table with {@code other}
     */
    <VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                    final Function<V, KO> foreignKeyExtractor,
                                    final ValueJoiner<V, VO, VR> joiner,
                                    final TableJoined<K, KO> tableJoined);

        /**
     * Join records of this {@code KTable} with another {@code KTable} using non-windowed inner join,
     * using the {@link TableJoined} instance for configuration of the {@link StreamPartitioner this table's
     * key serde} and {@link StreamPartitioner the other table's key serde}.
     * <p>
     * This is a foreign key join, where the joining key is determined by the {@code foreignKeyExtractor}.
     *
     * @param other               the other {@code KTable} to be joined with this {@code KTable}. Keyed by KO.
     * @param foreignKeyExtractor a {@link Function} that extracts the key (KO) from this table's value (V). If
the
     *                            result is null, the update is ignored as invalid.
     * @param joiner              a {@link ValueJoiner} that computes the join result for a pair of matching
records
     * @param tableJoined         a {@link TableJoined} used to configure partitioners and names of internal
topics and stores
     * @param materialized        a {@link Materialized} that describes how the {@link StateStore} for the
resulting {@code KTable}
     *                            should be materialized. Cannot be {@code null}
     * @param <VR>                the value type of the result {@code KTable}
     * @param <KO>                the key type of the other {@code KTable}
     * @param <VO>                the value type of the other {@code KTable}
     * @return a {@code KTable} that contains the result of joining this table with {@code other}
     */
    <VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                    final Function<V, KO> foreignKeyExtractor,
                                    final ValueJoiner<V, VO, VR> joiner,
                                    final TableJoined<K, KO> tableJoined,
                                    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

        /**
     * Join records of this {@code KTable} with another {@code KTable} using non-windowed left join,
     * using the {@link TableJoined} instance for configuration of the {@link StreamPartitioner this table's
     * key serde} and {@link StreamPartitioner the other table's key serde}.
     * <p>
     * This is a foreign key join, where the joining key is determined by the {@code foreignKeyExtractor}.
     *
     * @param other               the other {@code KTable} to be joined with this {@code KTable}. Keyed by KO.
     * @param foreignKeyExtractor a {@link Function} that extracts the key (KO) from this table's value (V) If
the
     *                            result is null, the update is ignored as invalid.
     * @param joiner              a {@link ValueJoiner} that computes the join result for a pair of matching
records
     * @param tableJoined         a {@link TableJoined} used to configure partitioners and names of internal
topics and stores
     * @param <VR>                the value type of the result {@code KTable}
     * @param <KO>                the key type of the other {@code KTable}
     * @param <VO>                the value type of the other {@code KTable}
     * @return a {@code KTable} that contains the result of joining this table with {@code other}
     */
    <VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                        final Function<V, KO> foreignKeyExtractor,
                                        final ValueJoiner<V, VO, VR> joiner,
                                        final TableJoined<K, KO> tableJoined);

        /**
     * Join records of this {@code KTable} with another {@code KTable} using non-windowed left join,
     * using the {@link TableJoined} instance for configuration of the {@link StreamPartitioner this table's
     * key serde} and {@link StreamPartitioner the other table's key serde}.
     * <p>
     * This is a foreign key join, where the joining key is determined by the {@code foreignKeyExtractor}.
     *
     * @param other               the other {@code KTable} to be joined with this {@code KTable}. Keyed by KO.
     * @param foreignKeyExtractor a {@link Function} that extracts the key (KO) from this table's value (V) If
the
     *                            result is null, the update is ignored as invalid.
     * @param joiner              a {@link ValueJoiner} that computes the join result for a pair of matching
records
     * @param tableJoined         a {@link TableJoined} used to configure partitioners and names of internal
topics and stores
     * @param materialized        a {@link Materialized} that describes how the {@link StateStore} for the
resulting {@code KTable}
     *                            should be materialized. Cannot be {@code null}
     * @param <VR>                the value type of the result {@code KTable}
     * @param <KO>                the key type of the other {@code KTable}
     * @param <VO>                the value type of the other {@code KTable}
     * @return a {@code KTable} that contains the result of joining this table with {@code other}
     */
    <VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                         final Function<V, KO> foreignKeyExtractor,
                                         final ValueJoiner<V, VO, VR> joiner,
                                         final TableJoined<K, KO> tableJoined,
                                         final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);
```

`TableJoined` itself is as follows. Note that the custom partitioners have type `StreamPartitioner<K, Void>` and `StreamPartitioner<KO, Void>` , rather than `StreamPartitioner<K, V>` and `StreamPartitioner<KO, VO>` as the original tables themselves use, in order to emphasize to the user that the custom partitioners must assign partitions based only on message keys and not values.

```
/**
 * The {@code TableJoined} class represents optional params that can be passed to
 * {@link KTable#join(KTable, Function, ValueJoiner, TableJoined) KTable#join(KTable,Function,...)} and
 * {@link KTable#leftJoin(KTable, Function, ValueJoiner, TableJoined) KTable#leftJoin(KTable,Function,...)}
 * operations, for foreign key joins.
 * @param <K>   this key type ; key type for the left (primary) table
 * @param <KO>  other key type ; key type for the right (foreign key) table
 */
public class TableJoined<K, KO> implements NamedOperation<TableJoined<K, KO>> {

        [...]

    /**
     * Create an instance of {@code TableJoined} with partitioner and otherPartitioner {@link
StreamPartitioner} instances.
     * {@code null} values are accepted and will result in the default partitioner being used.
     *
     * @param partitioner      a {@link StreamPartitioner} that specifies the partitioning strategy for the
left (primary)
     *                         table of the foreign key join. The partitioning strategy must depend only on the
message key
     *                         and not the message value. If {@code null} the default partitioner will be used.
     * @param otherPartitioner a {@link StreamPartitioner} that specifies the partitioning strategy for the
right (foreign
     *                         key) table of the foreign key join. The partitioning strategy must depend only
on the message
     *                         key and not the message value. If {@code null} the default partitioner will be
used.
     * @param <K>              this key type ; key type for the left (primary) table
     * @param <KO>             other key type ; key type for the right (foreign key) table
     * @return new {@code TableJoined} instance with the provided partitioners
     */
    public static <K, KO> TableJoined<K, KO> with(final StreamPartitioner<K, Void> partitioner,
                                                  final StreamPartitioner<KO, Void> otherPartitioner) {
        return new TableJoined<>(partitioner, otherPartitioner, null);
    }

    /**
     * Create an instance of {@code TableJoined} with base name for all components of the join, including
internal topics
     * created to complete the join.
     *
     * @param name the name used as the base for naming components of the join including internal topics
     * @param <K>  this key type ; key type for the left (primary) table
     * @param <KO> other key type ; key type for the right (foreign key) table
     * @return new {@code TableJoined} instance configured with the name
     *
     */
    public static <K, KO> TableJoined<K, KO> as(final String name) {
        return new TableJoined<>(null, null, name);
    }

    /**
     * Set the custom {@link StreamPartitioner} to be used. Null values are accepted and will result in the
     * default partitioner being used.
     *
     * @param partitioner the {@link StreamPartitioner} that specifies the partitioning strategy for the left
(primary)
     *                    table of the foreign key join. The partitioning strategy must depend only on the
message key
     *                    and not the message value. If {@code null} the default partitioner will be used.
     * @return new {@code TableJoined} instance configured with the {@code partitioner}
     */
    public TableJoined<K, KO> withPartitioner(final StreamPartitioner<K, Void> partitioner) {
        return new TableJoined<>(partitioner, otherPartitioner, name);
    }

    /**
     * Set the custom other {@link StreamPartitioner} to be used. Null values are accepted and will result
     * in the default partitioner being used.
     *
     * @param otherPartitioner the {@link StreamPartitioner} that specifies the partitioning strategy for the
right (foreign
     *                         key) table of the foreign key join. The partitioning strategy must depend only
on the message
     *                         key and not the message value. If {@code null} the default partitioner will be
used.
     * @return new {@code TableJoined} instance configured with the {@code otherPartitioner}
     */
    public TableJoined<K, KO> withOtherPartitioner(final StreamPartitioner<KO, Void> otherPartitioner) {
        return new TableJoined<>(partitioner, otherPartitioner, name);
    }

    /**
     * Set the base name used for all components of the join, including internal topics
     * created to complete the join.
     *
     * @param name the name used as the base for naming components of the join including internal topics
     * @return new {@code TableJoined} instance configured with the {@code name}
     */
    @Override
    public TableJoined<K, KO> withName(final String name) {
        return new TableJoined<>(partitioner, otherPartitioner, name);
    }
}
```

Proposed Changes

The new FK join interfaces shown above, along with the new `TableJoined` class, will be added. Existing methods will continue to use the default partitioner for subscription and response topics. Existing methods which accept `Named` will be marked for deprecation.

The new partitioners will be used by the FK join implementation as follows:

- `otherPartitioner` will be used when creating the subscription topic, to ensure copartitioning of the subscription topic and the foreign-key table.
- `thisPartitioner` will be used when creating the response topic, to ensure copartitioning of the response topic and the primary table.

In doing so, FK joins will be supported for source tables that do not use the default partitioner. It is up to the user to ensure copartitioning of FK join subscription and response topics by passing the correct partitioners.

Compatibility, Deprecation, and Migration Plan

Existing FK join interfaces (without custom partitioners) will continue to use the default partitioner for subscription and response topics. Existing methods which accept `Named` will be marked for deprecation (to be removed in the next major release, likely 4.0).

Rejected Alternatives

N/A