KIP-669: Preserve Source Partition in Kafka Streams from context

Status

Current state: *"Under Discussion"*

Discussion thread: NA

JIRA: *KAFKA-10448*

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

Motivation

In most use cases of Kafka streams where there is need for preserving partitions from source to sink topic, currently the approach relies on default partitioner which is a pure play based on key and value. While this approach works in most cases where the partitioner used for the producer to source topic and stream partitioner follow the same principle of deriving partition based on key, it does not work in use cases where the partitioner used for producer is either unknown or could not be easily mapped to keys. In addition, as source partition is already available from the Processor Context in sink node, it would only make sense to use this if we would like to preserve partitions from source to sink from context rather than re-evaluating it again.

Here is an example of how this is a drawback

Imagine we have a producer application which is reading from a database and publishing data into source topics. Assume there are are different instances of producer application, each reading from different partition of a database table. In order to maintain processing sequence each of the producer instance decide to use only a subset of partitions, but the keys written to source topic do not match with partitioning decision made by the producer as producer is deciding partitions based on some other "partitioning key" available in the source database. In order to propagate the messages now from the source topic while still maintaining the same sequence of messages between source and sink, the stream-partitioner now needs the similar logic like the producer. As the original "partitioning key" is not available in the message key-value pair, Stream-partitioner now does not have the ability to mimic same logic. As a consequence, messages in sink topic are now interleaving and the ordering guarantee is lost.

This is only an example, however there are other benefits of using context as it minimizes reprocessing of partition logic at sink node.

To fill this gap, this KIP proposes to allow using context partition from source topic as is as an option, so such use cases can benefit from it.

Public Interfaces

Add a new constructor for SinkNode.java as below

org.apache.kafka.streams.processor.internals.SinkNode

public class SinkNode {

/*Existing Code*/

....

SinkNode(final String name, final TopicNameExtractor<K, V> topicExtractor, final Serializer<K> keySerializer, final Serializer<V> valSerializer, final boolean useContextPartition) {}

/*Existing Code*/

....

```
}
```

Add additional methods for Topology.java as below

org/apache/kafka/streams/Topology.java

Topology.java

```
public class Topology {

  // Existing code
  //

  // New method signature as below
  /**
     * Add a new sink that forwards records from upstream parent processor and/or source nodes to the named
Kafka topic,
     * using the context partitioner
     * The sink will use the {@link StreamsConfig#DEFAULT_KEY_SERDE_CLASS_CONFIG default key serializer} and
     * {@link StreamsConfig#DEFAULT_VALUE_SERDE_CLASS_CONFIG default value serializer} specified in the
     * {@link StreamsConfig stream configuration}.
     * <p>
     * The sink will use the paritition based on the context {@link ProcessorContext} of the message being
processed
     * Such control is often useful with topologies that do not have the knowledge of how partitioning is used
when messages
     * were originally published into source topic. This could also be used as an alternate approach to using
default partitioner
     * without the need for re-evaluating partitions. Kafka's default partitioning logic will be used
automatically in the event
     * context partition is not used.
     *
     * @param name the unique name of the sink
     * @param topic the name of the Kafka topic to which this sink should write its records
     * @param useContextPartition the function that should be used to determine the partition for each record
processed by the sink
     * @param parentNames the name of one or more source or processor nodes whose output records this sink
should consume
     * and write to its topic
     * @return itself
     * @throws TopologyException if parent processor is not added yet, or if this processor's name is equal to
the parent's name
     * @see #addSink(String, String, Serializer, Serializer, StreamPartitioner, String...)
     */
    public synchronized <K, V> Topology addSink(final String name,
                                                final String topic,
                                                final boolean useContextPartition,
                                                final String... parentNames) {
        internalTopologyBuilder.addSink(name, topic, null, null, useContextPartition, parentNames);
        return this;
    }


    /**
     * Add a new sink that forwards records from upstream parent processor and/or source nodes to the named
Kafka topic.
     * The sink will use the specified key and value serializers, and would preserve the partition from source
     *
     * @param name the unique name of the sink
     * @param topic the name of the Kafka topic to which this sink should write its records
     * @param keySerializer the {@link Serializer key serializer} used when consuming records; may be null if
the sink
     * should use the {@link StreamsConfig#DEFAULT_KEY_SERDE_CLASS_CONFIG default key serializer} specified in
the
     * {@link StreamsConfig stream configuration}
     * @param valueSerializer the {@link Serializer value serializer} used when consuming records; may be null
```

```
if the sink
     * should use the {@link StreamsConfig#DEFAULT_VALUE_SERDE_CLASS_CONFIG default value serializer} specified
in the
     * {@link StreamsConfig stream configuration}
     * @param useContextPartition the boolean that determines whether to use context partition or default
partitioner
     * @param parentNames the name of one or more source or processor nodes whose output records this sink
should consume
     * and write to its topic
     * @return itself
     * @throws TopologyException if parent processor is not added yet, or if this processor's name is equal to
the parent's name
     * @see #addSink(String, String, String...)
     * @see #addSink(String, String, StreamPartitioner, String...)
     * @see #addSink(String, String, Serializer, Serializer, String...)
     */
    public synchronized <K, V> Topology addSink(final String name,
                                                final String topic,
                                                final Serializer<K> keySerializer,
                                                final Serializer<V> valueSerializer,
                                                final boolean useContextPartition,
                                                final String... parentNames) {
        internalTopologyBuilder.addSink(name, topic, keySerializer, valueSerializer, useContextPartition,
parentNames);
        return this;
    }

     /**
     * Add a new sink that forwards records from upstream parent processor and/or source nodes to Kafka topics
based on {@code topicExtractor}.
     * The topics that it may ever send to should be pre-created.
     * The sink will use the specified key and value serializers.
     *
     * @param name               the unique name of the sink
     * @param topicExtractor     the extractor to determine the name of the Kafka topic to which this sink
should write for each record
     * @param keySerializer      the {@link Serializer key serializer} used when consuming records; may be null
if the sink
     *                           should use the {@link StreamsConfig#DEFAULT_KEY_SERDE_CLASS_CONFIG default key
serializer} specified in the
     *                           {@link StreamsConfig stream configuration}
     * @param valueSerializer    the {@link Serializer value serializer} used when consuming records; may be
null if the sink
     *                           should use the {@link StreamsConfig#DEFAULT_VALUE_SERDE_CLASS_CONFIG default
value serializer} specified in the
     *                           {@link StreamsConfig stream configuration}
     * @param useContextPartition  the {@link ProcessorContext context.partition} used when producing messages
to sink topic
     * @param parentNames        the name of one or more source or processor nodes whose output records this
sink should consume
     *                           and dynamically write to topics
     * @return                   itself
     * @throws TopologyException if parent processor is not added yet, or if this processor's name is equal to
the parent's name
     * @see #addSink(String, String, Serializer, Serializer, boolean, String...)
     */
    public synchronized <K, V> Topology addSink(final String name,
                                                final TopicNameExtractor<K, V> topicExtractor,
                                                final Serializer<K> keySerializer,
                                                final Serializer<V> valueSerializer,
                                                final boolean useContextPartition,
                                                final String... parentNames) {
        internalTopologyBuilder.addSink(name, topicExtractor, keySerializer, valueSerializer,
useContextPartition, parentNames);
        return this;
    }
```

Proposed Changes

- Add a new boolean instance variable (useContextPartition) to SinkNode to keep track of whether to preserve partition from context.
- Existing StreamPartitioner in SinkNode will be made null when context partition is enabled
- Add appropriate methods in Topology.java and InternalToplogyBuilder.java to overload current addSink methods to accept whether to use context partition
- SinkNode process method will be changed to use context partition directly using the alternate send method of RecordCollector if useContextPartition is true

Compatibility, Deprecation, and Migration Plan

- There should not be any impact to existing users as all existing methods will be kept as is.

Rejected Alternatives

Adding options in Stream Partitioner to use message headers to derive partition optionally. Record collector could then provide message headers to derive partition. Using context partition however seemed straight forward to preserve partitions rather than using message headers.