Optimizing Kafka Streams Topologies

Optimization FAQ

While details of Optimization are in the following section, this section serves to give quick guidance on using Optimizations in a Kafka Streams application.

What are Optimizations?

Kafka Streams optimizations are an attempt to automatically make Kafka Streams applications more efficient by reorganizing a topology based on the inital construction of the Kafka Streams application. Right now there are two possible optimizations, reusing the source topic as a changelog topic for a `KTable` created directly from an input topic. The second optimization is merging multiple repartiton topics into one repartition topic when there mulitple grouping or join operations after a key-changing operation.

Should I enable optimizations for my existing application?

The answer is it depends. If you have source `KTable` processors or you change keys followed by either grouping/aggregation operations or joins then the answer is yes. For now, if your application does not have either of those two outlined scenarios, you don't need to enable optimizations but it is safe to do so as it won't have any affect on your topology.

How Do I enable optimizations for my existing applications?

To enable optimizations, you need to do two things. First, add this line to your properties `properties.setProperty(StreamsConfig.TOPOLOGY_OPTIMIZATION, StreamsConfig.OPTIMIZE);`. Second, when constructing your `KafkaStreams` instance, you'll need to pass your configuration properties when building your topology by using the overloaded `StreamsBuilder.build(Properties)` method. For example, `KafkaStreams myStream = new KafkaStreams(streamsBuilder.build(properties), properties)`.

Optimizations are disabled by default, so if you don't want to enable optmizations, you don't have to take any action.

Should I enable optimizations for my new application?

The same guidance applies for those with existing applications, with the exception you don't need to use the Streams Reset tool. Additionally if you start with optimization enabled, then it is safe to do a rolling deployment.

I've enabled optimizations for my existing application, are there any additional steps?

If you have an existing application, a rolling upgrade is not possible. You'll need to take these steps:

1. Stop all your application instances.
2. Update the configuration on all your application instances as indicated above.
3. Run the Streams Reset tool to reprocess your data. While not strictly required, there may be unprocessed data in the merged reparition topics, so reprocessing data is a safeguard against missing any records. If your business needs don't require processing all records you may be able to skip the reprocessing step.

Please note that after the initial redeployment, your application will not go under any additional optimization changes. Also if you add processors that are optimizable after the initial redeployment, you don't need to take any additional steps as the repartition topics will already be merged.

Optimization Details

In Confluent 5.1, Kafka Streams added a new feature that enables users to choose to allow the Kafka Streams framework to optimize topology by setting the config flag `StreamsConfig.TOPOLOGY_OPTIMIZATION` to `StreamsConfig.OPTIMIZE`. The default setting for

`StreamsConfig.TOPOLOGY_OPTIMIZATION` is `StreamsConfig.NO_OPTIMIZATION`, so if you don't want enable optimizations then there is nothing further you need to do. Optimizations are available on the DSL only, as the Processor API already gives users the flexibility required to enable their own optimizations, so an optimization configuration isn't needed for the Processor API.

Before we discuss the available optimizations, here's some context for why we added this optimization feature. Prior to CP 5.1/AK 2.1, when you used the DSL to build a topology, Kafka Streams constructed the parts of the physical plan of the topology immediately with each call to the DSL. By starting to construct a physical plan immediately, any opportunity to make changes for a more efficient topology is lost.

Now, when building a topology with the DSL, each call is captured in an intermediate representation or logical plan of the topology, represented as a directed acyclic graph (DAG). The physical plan for the topology isn't started until the `StreamsBuilder.build(properties)` call is executed. During the conversion of the logical plan to the physical plan, any possible optimizations are applied. Using the no-arg `StreamsBuilder.build()` method, Kafka Streams doesn't make any optimizations to the topology.

Currently, there are two optimizations that Kafka Streams performs when enabled:

1. The source `KTable` re-uses the source topic as the changelog topic.
2. When possible, Kafka Streams collapses multiple repartition topics into a single repartition topic.

The first optimization is straightforward and doesn't require further discussion. The second optimization is a little more involved and requires some explanation.

Consider the following toplogy:

```
final KStream<String, String> sourceStream = builder.stream(inputTopic, Consumed.with(Serdes.String(), Serdes.String())
);

// Because we are changing the key, a "needs repartitioning" flag is set.
// The repartitioning will be performed only when a subsequent operation truly needs it.
final KStream<String, String> mappedStream = sourceStream.selectKey((k, v) -> keyFunction.apply(v));

// triggers the first repartitioning grouping by the changed key
mappedStream.groupByKey().windowedBy(...).aggregate(...)...;

// triggers the second repartitioning grouping by the changed key again
KStream<String, Long> countStream = mappedStream.groupByKey().count(...).toStream();

// triggers the third reparition for joining the mappedStream with the changed key
KStream<String, String> joinedStream = mappedStream.join(countStream.....);


config.setProperty(StreamsConfig.APPLICATION_ID_CONFIG, "SomeStreamsApplication");
config.setProperty(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,"Some IP Address");

// tell Kafka Streams to optimize the topology
config.setProperty(StreamsConfig.TOPOLOGY_OPTIMIZATION, StreamsConfig.OPTIMIZE);

// Since we've configured Streams to use optimizations, the topology is optimized during the build
// And because optimizations are enabled, the resulting topology will no longer need to perform
// three explicit repartitioning steps, but only one.
final Topology topology = builder.build(config);
final KafkaStreams streams = new KafkaStreams(topology, config);

streams.start();
```

Since we're changing the key with the `KStream.selectKey()` method, a boolean flag indicating a reparititon may be required. Note that the repartition doesn't happen automatically after changing the key in a Kafka Streams application. A repartition is performed only when an operation requires the potentially changed key.

From the previous example, Kafka Streams creates three repartition topics, since we're performing three operations – two `KStream.groupByKey()` and one `KStream.join()` – requiring the key after we have potentially modified the key on the original stream. It's worth noting at this point that using *KStream.map*, *KStream.transform*, and *KStream.flatMap* always sets the "needs repartition" flag even if you don't physically change the key, so if you know that you don't need to change the key, use the `KStream.xxxValues` options instead.

But all three repartition topics will contain virtually the same data, so two of the three repartition topics are redundant. With optimizations enabled, Kafka Streams removes the repartitioning from the two `KStream.groupByKey()` calls and the single `KStream.join()` call and uses *one* repartition topic immediately after the `KStream.selectKey()`, which reduces the repartition topic load from three to one.

Finally, due to the scope of the topology change with respect to repartition topics, when enabling optimizations, a rolling upgrade isn't possible. You must stop all instances of your application. Since there could be a small amount of data left unprocessed in the merged repartition topics,

we recommend using the Streams Reset tool to cleanup your previous application, and start the new one from the beginning to reprocess your data. Otherwise, there may be a small amount of missing records.

Please report any inaccuracies on this page or suggest an edit.

3 Votes

Last updated on Sep 10, 2019.