Item 62
git_comments:

1. * Licensed to the Apache Software Foundation (ASF) under one or more * contributor license agreements. See the NOTICE file distributed with * this work for additional information regarding copyright ownership. * The ASF licenses this file to You under the Apache License, Version 2.0 * (the "License"); you may not use this file except in compliance with * the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.

git_commits:

1. **summary:** KAFKA-6782: solved the bug of restoration of aborted messages for GlobalStateStore and KGlobalTable (#4900)
   message: KAFKA-6782: solved the bug of restoration of aborted messages for GlobalStateStore and KGlobalTable (#4900) Reviewer: Matthias J. Sax <matthias@confluent.io>, Bill Bejeck <bill@confluent.io>, Guozhang Wang <guozhang@confluent.io>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** KAFKA-6782: solved the bug of restoration of aborted messages for GlobalStateStore and KGlobalTable
   body:

github_pulls_comments:

1. \cc @bbejeck @vvcephei
2. @Gitomain what is the status of this PR -- seems there are still couple of comments you did not address yet.
3. Hello, I have deleted all the .gitignore files and added a new test GlobalKTableEOSIntegrationTest to test the EOS case independently.
4. There is a recent commit that moves which package `Consumed` belongs, causing the jenkins to fail: ``` 17:02:22 /home/jenkins/jenkins-slave/workspace/kafka-pr-jdk8-scala2.11/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java:26: error: cannot find symbol 17:02:22 import org.apache.kafka.streams.Consumed; 17:02:22 ^ 17:02:22 symbol: class Consumed 17:02:22 location: package org.apache.kafka.streams 17:02:25 /home/jenkins/jenkins-slave/workspace/kafka-pr-jdk8-scala2.11/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java:114: error: cannot find symbol 17:02:25 globalTable = builder.globalTable(globalTableTopic, Consumed.with(Serdes.Long(), Serdes.String()), 17:02:25 ^ 17:02:25 symbol: variable Consumed 17:02:25 location: class GlobalKTableEOSIntegrationTest 17:02:25 /home/jenkins/jenkins-slave/workspace/kafka-pr-jdk8-scala2.11/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java:118: error: cannot find symbol 17:02:25 final Consumed<String, Long> stringLongConsumed = Consumed.with(Serdes.String(), Serdes.Long()); 17:02:25 ^ 17:02:25 symbol: class Consumed 17:02:25 location: class GlobalKTableEOSIntegrationTest 17:02:25 /home/jenkins/jenkins-slave/workspace/kafka-pr-jdk8-scala2.11/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java:118: error: cannot find symbol 17:02:25 final Consumed<String, Long> stringLongConsumed = Consumed.with(Serdes.String(), Serdes.Long()); ``` Could you rebase your branch on latest trunk head and merge any conflicts?
5. For the motivation of extracting the EOS test into a separate class, I think both the EOS enabled and the EOS disabled cases should be tested. And I found it was a little messy to test the 2 cases in the same class. So I extracted the EOS test into a separate class.
6. @Gitomain why include `log4j` files for connect and tools?
7. @Gitomain Code freeze for 2.0 release is on Tuesday (6/12). Can you finish this PR so we can get it into the release? (If not, I might just take it and apply the last changes myself to merged it.) Please let us know.

8. Hello @mjsax , I have done some modification. But I have no time to extract shared code into an abstract class before 6/12. I think we can finish this PR now.
9. The deadline was pushed out by one day. I think we can merge as-is. The code rewrites to Java8 are not too important.
10. Thanks for the fix @Gitomain Merged to `trunk` and cherry-picked to `2.0`, `1.1`, `1.0`, and `0.11.0`.

github_pulls_reviews:

1. Please avoid unnecessary reformatting -- there is more in this PR -- guess, it's a matter of IDE settings?
2. I think it would be better, to have two test -- one with EOS enabled and one without.
3. Why this change? We don't need the variable `topology`?
4. nit: in tests, it better to just declare `throws Exception` -- it does not provide value to spell out the different types in test and introduces "noise" I realized, that the existing code also does declare different types of exception. Would you mind to clean this up, too?
5. nit: add `final` (we try to use `final` whenever possible)
6. nit: formatting -- parameters should be aligned
7. nit: add `final`
8. nit: exceptions
9. nit: the name is a little miss leading -- aborted message should not be restored... It also seems, this test only checks if the restore terminates -- I am wondering, if we should also write committed data into the topic and check that the messages got restored? Not sure if necessary.
10. Yes, it's a matter of IDE settings, we don't need these chages.
11. nit: formatting - align parameters
12. Are those changes intended? Or are they IDE enforced?
13. I think it's a problem of IDE
14. nit: formatting
15. with Java 8 support, I think we can remove generics here
16. please address this comment, too
17. do we need a `flush()` here? `abortTransaction()` is a sync call
18. this line can be removed
19. do we need the timeout? `waitForCondition` already applies a timeout
20. this method should enable EOS by default for this class, shouldn't it?
21. we can merge this method into the one above as we always want EOS enabled?
22. Just one meta-comment, now that Streams supports Java 8 do we want to covert the use of anonymous class usage here and other places to lambda expressions? This comment is very opinionated, and this PR has been going on for a while now, so I wouldn't hold up merging for this.
23. Hello, I can't understand how can we remove generics here, so I didn't modify this part.
24. Thanks for remind, it's done
25. Hello, I know what you mean. But I have seen that all the test codes are still in this style now, I prefer to leave it as it is, what do you think ?
26. Java8 should be able to infer the type and Java7 was dropped recently. So we want to update the code to Java8 incrementally. You should be able to change this to: ``` Materialized.as(globalStore) ```
27. As mentioned in the other comment. Java7 was just dropped recently and thus most code is still Java7. We want to incrementally rewrite to Java8. Thus, feel free to update the code accordingly.
28. @Gitomain We can add `*.class` but we need to keep `classes` as some folders are named like this. Can you update the PR? Otherwise I can fix during merging.

jira_issues:

1. **summary:** GlobalKTable GlobalStateStore never finishes restoring when consuming aborted messages **description:** Same problem with https://issues.apache.org/jira/browse/KAFKA-6190, but his solution which is below, works for the succeed transactional messages. But when there are aborted messages, it will be in infinite loop. Here is his proposition : {code:java} while (offset < highWatermark) { final ConsumerRecords<byte[], byte[]> records = consumer.poll(100); for (ConsumerRecord<byte[], byte[]> record : records) { if (record.key() != null) { stateRestoreCallback.restore(record.key(), record.value()); } offset = consumer.position(topicPartition); } }{code} Concretely, when the consumer consume a set of aborted messages, it polls 0 records, and the code 'offset = consumer.position(topicPartition)' doesn't have any opportunity to execute. So I propose to move the code 'offset = consumer.position(topicPartition)' outside of the cycle to guarantee that event if no records are polled, the offset can always be updated. {code:java} while (offset < highWatermark) { final ConsumerRecords<byte[], byte[]> records = consumer.poll(100); for (ConsumerRecord<byte[], byte[]> record : records) { if (record.key() != null) {

stateRestoreCallback.restore(record.key(), record.value()); } } offset = consumer.position(topicPartition); } {code}

jira_issues_comments:

1. Thanks for reporting this issue. Make sense to me -- feel free to open an PR. I would assume that it affects GlobalKTables, too? Thus, we should add tests for both cases.
2. Yes, I think GlobalKTables will have the same problem. I'm trying to add PR and test for it.
3. Gitomain opened a new pull request #4900: KAFKA-6782: solved the bug of restoration of aborted messages for GlobalStateStore and KGlobalTable URL: https://github.com/apache/kafka/pull/4900 ---------- ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org
4. mjsax closed pull request #4900: KAFKA-6782: solved the bug of restoration of aborted messages for GlobalStateStore and KGlobalTable URL: https://github.com/apache/kafka/pull/4900 This is a PR merged from a forked repository. As GitHub hides the original diff on merge, it is displayed below for the sake of provenance: As this is a foreign pull request (from a fork), the diff is supplied below (as it won't show otherwise due to GitHub magic): diff --git a/.gitignore b/.gitignore index 04f8feed0ad..fe191eed44b 100644 --- a/.gitignore +++ b/.gitignore @@ -1,5 +1,6 @@ dist *classes +*.class target/ build/ build_eclipse/ diff --git a/kafka b/kafka new file mode 160000 index 00000000000..cc43e77bbbf --- /dev/null +++ b/kafka @@ -0,0 +1 @@ +Subproject commit cc43e77bbbfad71883011186de55603c936cbcd1 diff --git a/streams/src/main/java/org/apache/kafka/streams/processor/internals/GlobalStateManagerImpl.java b/streams/src/main/java/org/apache/kafka/streams/processor/internals/GlobalStateManagerImpl.java index e8ec5e9fe5f..96064b6d4ad 100644 --- a/streams/src/main/java/org/apache/kafka/streams/processor/internals/GlobalStateManagerImpl.java +++ b/streams/src/main/java/org/apache/kafka/streams/processor/internals/GlobalStateManagerImpl.java @@ -268,8 +268,8 @@ private void restoreState(final StateRestoreCallback stateRestoreCallback, if (record.key() != null) { restoreRecords.add(KeyValue.pair(record.key(), record.value())); } - offset = globalConsumer.position(topicPartition); } + offset = globalConsumer.position(topicPartition); stateRestoreAdapter.restoreAll(restoreRecords); stateRestoreListener.onBatchRestored(topicPartition, storeName, offset, restoreRecords.size()); restoreCount += restoreRecords.size(); diff --git a/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java b/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java new file mode 100644 index 00000000000..f7c0e55c05e --- /dev/null +++ b/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableEOSIntegrationTest.java @@ -0,0 +1,390 @@ +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ +package org.apache.kafka.streams.integration; + +import kafka.utils.MockTime; +import org.apache.kafka.clients.consumer.ConsumerConfig; +import org.apache.kafka.clients.producer.ProducerConfig; +import org.apache.kafka.common.serialization.LongSerializer; +import org.apache.kafka.common.serialization.Serdes; +import org.apache.kafka.common.serialization.StringSerializer; +import org.apache.kafka.common.utils.Bytes; +import org.apache.kafka.streams.kstream.Consumed; +import org.apache.kafka.streams.KafkaStreams; +import org.apache.kafka.streams.KeyValue; +import org.apache.kafka.streams.StreamsBuilder; +import org.apache.kafka.streams.StreamsConfig; +import org.apache.kafka.streams.errors.InvalidStateStoreException; +import org.apache.kafka.streams.integration.utils.EmbeddedKafkaCluster; +import org.apache.kafka.streams.integration.utils.IntegrationTestUtils; +import org.apache.kafka.streams.kstream.ForeachAction; +import org.apache.kafka.streams.kstream.GlobalKTable; +import org.apache.kafka.streams.kstream.KStream; +import org.apache.kafka.streams.kstream.KeyValueMapper; +import org.apache.kafka.streams.kstream.Materialized; +import org.apache.kafka.streams.kstream.ValueJoiner; +import org.apache.kafka.streams.state.KeyValueStore; +import org.apache.kafka.streams.state.QueryableStoreTypes; +import org.apache.kafka.streams.state.ReadOnlyKeyValueStore; +import

org.apache.kafka.test.IntegrationTest; +import org.apache.kafka.test.TestCondition; +import org.apache.kafka.test.TestUtils; +import org.junit.After; +import org.junit.Before; +import org.junit.ClassRule; +import org.junit.Test; +import org.junit.experimental.categories.Category; + +import java.io.IOException; +import java.util.Arrays; +import java.util.HashMap; +import java.util.Iterator; +import java.util.Map; +import java.util.Properties; + +@Category({IntegrationTest.class}) +public class GlobalKTableEOSIntegrationTest { + private static final int NUM_BROKERS = 1; + private static final Properties BROKER_CONFIG; + static { + BROKER_CONFIG = new Properties(); + BROKER_CONFIG.put("transaction.state.log.replication.factor", (short) 1); + BROKER_CONFIG.put("transaction.state.log.min.isr", 1); + } + + @ClassRule + public static final EmbeddedKafkaCluster CLUSTER = + new EmbeddedKafkaCluster(NUM_BROKERS, BROKER_CONFIG); + + private static volatile int testNo = 0; + private final MockTime mockTime = CLUSTER.time; + private final KeyValueMapper<String, Long, Long> keyMapper = new KeyValueMapper<String, Long, Long>() { + @Override + public Long apply(final String key, final Long value) { + return value; + } + }; + private final ValueJoiner<Long, String, String> joiner = new ValueJoiner<Long, String, String>() { + @Override + public String apply(final Long value1, final String value2) { + return value1 + "+" + value2; + } + }; + private final String globalStore = "globalStore"; + private final Map<String, String> results = new HashMap<>(); + private StreamsBuilder builder; + private Properties streamsConfiguration; + private KafkaStreams kafkaStreams; + private String globalTableTopic; + private String streamTopic; + private GlobalKTable<Long, String> globalTable; + private KStream<String, Long> stream; + private ForeachAction<String, String> foreachAction; + + @Before + public void before() throws InterruptedException { + testNo++; + builder = new StreamsBuilder(); + createTopics(); + streamsConfiguration = new Properties(); + final String applicationId = "globalTableTopic-table-eos-test-" + testNo; + streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, applicationId); + streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, CLUSTER.bootstrapServers()); + streamsConfiguration.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); + streamsConfiguration.put(StreamsConfig.STATE_DIR_CONFIG, TestUtils.tempDirectory().getPath()); + streamsConfiguration.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0); + streamsConfiguration.put(IntegrationTestUtils.INTERNAL_LEAVE_GROUP_ON_CLOSE, true); + streamsConfiguration.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 100); + streamsConfiguration.put(StreamsConfig.PROCESSING_GUARANTEE_CONFIG, "exactly_once"); + globalTable = builder.globalTable(globalTableTopic, Consumed.with(Serdes.Long(), Serdes.String()), + Materialized.<Long, String, KeyValueStore<Bytes, byte[]>>as(globalStore) + .withKeySerde(Serdes.Long()) + .withValueSerde(Serdes.String())); + final Consumed<String, Long> stringLongConsumed = Consumed.with(Serdes.String(), Serdes.Long()); + stream = builder.stream(streamTopic, stringLongConsumed); + foreachAction = new ForeachAction<String, String>() { + @Override + public void apply(final String key, final String value) { + results.put(key, value); + } + }; + } + + @After + public void whenShuttingDown() throws IOException { + if (kafkaStreams != null) { + kafkaStreams.close(); + } + IntegrationTestUtils.purgeLocalStreamsState(streamsConfiguration); + } + + @Test + public void shouldKStreamGlobalKTableLeftJoin() throws Exception { + final KStream<String, String> streamTableJoin = stream.leftJoin(globalTable, keyMapper, joiner); + streamTableJoin.foreach(foreachAction); + produceInitialGlobalTableValues(); + startStreams(); + produceTopicValues(streamTopic); + + final Map<String, String> expected = new HashMap<>(); + expected.put("a", "1+A"); + expected.put("b", "2+B"); + expected.put("c", "3+C"); + expected.put("d", "4+D"); + expected.put("e", "5+null"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return results.equals(expected); + } + }, 30000L, "waiting for initial values"); + + produceGlobalTableValues(); + + final ReadOnlyKeyValueStore<Long, String> replicatedStore = kafkaStreams.store(globalStore, QueryableStoreTypes.<Long, String>keyValueStore()); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return "J".equals(replicatedStore.get(5L)); + } + }, 30000, "waiting for data in replicated store"); + produceTopicValues(streamTopic); + + expected.put("a", "1+F"); + expected.put("b", "2+G"); + expected.put("c", "3+H"); + expected.put("d", "4+I"); + expected.put("e", "5+J"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return results.equals(expected); + } + }, 30000L, "waiting for final values"); + } + + @Test + public void shouldKStreamGlobalKTableJoin() throws Exception { + final KStream<String, String> streamTableJoin = stream.join(globalTable, keyMapper, joiner); + streamTableJoin.foreach(foreachAction); + produceInitialGlobalTableValues(); + startStreams(); + produceTopicValues(streamTopic); + + final Map<String, String> expected = new HashMap<>(); + expected.put("a", "1+A"); + expected.put("b", "2+B"); + expected.put("c", "3+C"); + expected.put("d", "4+D"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return results.equals(expected); + } + }, 30000L, "waiting for initial values"); + + produceGlobalTableValues(); + + final ReadOnlyKeyValueStore<Long, String> replicatedStore = kafkaStreams.store(globalStore, QueryableStoreTypes.<Long, String>keyValueStore()); +

TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return "J".equals(replicatedStore.get(5L)); + } + }, 30000, "waiting for data in replicated store"); + + produceTopicValues(streamTopic); + + expected.put("a", "1+F"); + expected.put("b", "2+G"); + expected.put("c", "3+H"); + expected.put("d", "4+I"); + expected.put("e", "5+J"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + return results.equals(expected); + } + }, 30000L, "waiting for final values"); + } + + @Test + public void shouldRestoreTransactionalMessages() throws Exception { + produceInitialGlobalTableValues(); + + startStreams(); + + final Map<Long, String> expected = new HashMap<>(); + expected.put(1L, "A"); + expected.put(2L, "B"); + expected.put(3L, "C"); + expected.put(4L, "D"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + ReadOnlyKeyValueStore<Long, String> store = null; + try { + store = kafkaStreams.store(globalStore, QueryableStoreTypes.<Long, String>keyValueStore()); + } catch (InvalidStateStoreException ex) { + return false; + } + Map<Long, String> result = new HashMap<>(); + Iterator<KeyValue<Long, String>> it = store.all(); + while (it.hasNext()) { + KeyValue<Long, String> kv = it.next(); + result.put(kv.key, kv.value); + } + return result.equals(expected); + } + }, 30000L, "waiting for initial values"); + } + + @Test + public void shouldNotRestoreAbortedMessages() throws Exception { + produceAbortedMessages(); + produceInitialGlobalTableValues(); + produceAbortedMessages(); + + startStreams(); + + final Map<Long, String> expected = new HashMap<>(); + expected.put(1L, "A"); + expected.put(2L, "B"); + expected.put(3L, "C"); + expected.put(4L, "D"); + + TestUtils.waitForCondition(new TestCondition() { + @Override + public boolean conditionMet() { + ReadOnlyKeyValueStore<Long, String> store = null; + try { + store = kafkaStreams.store(globalStore, QueryableStoreTypes.<Long, String>keyValueStore()); + } catch (InvalidStateStoreException ex) { + return false; + } + Map<Long, String> result = new HashMap<>(); + Iterator<KeyValue<Long, String>> it = store.all(); + while (it.hasNext()) { + KeyValue<Long, String> kv = it.next(); + result.put(kv.key, kv.value); + } + return result.equals(expected); + } + }, 30000L, "waiting for initial values"); + } + + private void createTopics() throws InterruptedException { + streamTopic = "stream-" + testNo; + globalTableTopic = "globalTable-" + testNo; + CLUSTER.createTopics(streamTopic); + CLUSTER.createTopic(globalTableTopic, 2, 1); + } + + private void startStreams() { + kafkaStreams = new KafkaStreams(builder.build(), streamsConfiguration); + kafkaStreams.start(); + } + + private void produceTopicValues(final String topic) throws Exception { + IntegrationTestUtils.produceKeyValuesSynchronously(+ topic, + Arrays.asList(+ new KeyValue<>("a", 1L), + new KeyValue<>("b", 2L), + new KeyValue<>("c", 3L), + new KeyValue<>("d", 4L), + new KeyValue<>("e", 5L)), + TestUtils.producerConfig(+ CLUSTER.bootstrapServers(), + StringSerializer.class, + LongSerializer.class, + new Properties()), + mockTime); + } + + private void produceAbortedMessages() throws Exception { + final Properties properties = new Properties(); + properties.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, "someid"); + properties.put(ProducerConfig.RETRIES_CONFIG, 1); + IntegrationTestUtils.produceAbortedKeyValuesSynchronouslyWithTimestamp(+ globalTableTopic, + Arrays.asList(+ new KeyValue<>(1L, "A"), + new KeyValue<>(2L, "B"), + new KeyValue<>(3L, "C"), + new KeyValue<>(4L, "D") +), + TestUtils.producerConfig(+ CLUSTER.bootstrapServers(), + LongSerializer.class, + StringSerializer.class, + properties), + mockTime.milliseconds()); + } + + private void produceInitialGlobalTableValues() throws Exception { + produceInitialGlobalTableValues(true); + } + + private void produceInitialGlobalTableValues(final boolean enableTransactions) throws Exception { + final Properties properties = new Properties(); + if (enableTransactions) { + properties.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, "someid"); + properties.put(ProducerConfig.RETRIES_CONFIG, 1); + } + IntegrationTestUtils.produceKeyValuesSynchronously(+ globalTableTopic, + Arrays.asList(+ new KeyValue<>(1L, "A"), + new KeyValue<>(2L, "B"), + new KeyValue<>(3L, "C"), + new KeyValue<>(4L, "D") +), + TestUtils.producerConfig(+ CLUSTER.bootstrapServers(), + LongSerializer.class, + StringSerializer.class, + properties), + mockTime, + enableTransactions); + } + + private void produceGlobalTableValues() throws Exception { + IntegrationTestUtils.produceKeyValuesSynchronously(+ globalTableTopic, + Arrays.asList(+ new KeyValue<>(1L, "F"), + new KeyValue<>(2L, "G"), + new KeyValue<>(3L, "H"), + new KeyValue<>(4L, "I"), + new KeyValue<>(5L, "J")), + TestUtils.producerConfig(+ CLUSTER.bootstrapServers(), + LongSerializer.class, + StringSerializer.class, + new Properties()), + mockTime); + } +} diff --git a/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableIntegrationTest.java b/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableIntegrationTest.java index 8c6a30a5972..900e65276ee 100644 --- a/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableIntegrationTest.java +++ b/streams/src/test/java/org/apache/kafka/streams/integration/GlobalKTableIntegrationTest.java @@ -18,7 +18,6 @@ import kafka.utils.MockTime; import org.apache.kafka.clients.consumer.ConsumerConfig; -import org.apache.kafka.clients.producer.ProducerConfig; import org.apache.kafka.common.serialization.LongSerializer; import org.apache.kafka.common.serialization.Serdes;

import org.apache.kafka.common.serialization.StringSerializer; @@ -28,7 +27,6 @@ import org.apache.kafka.streams.KeyValue; import org.apache.kafka.streams.StreamsBuilder; import org.apache.kafka.streams.StreamsConfig; -import org.apache.kafka.streams.errors.InvalidStateStoreException; import org.apache.kafka.streams.integration.utils.EmbeddedKafkaCluster; import org.apache.kafka.streams.integration.utils.IntegrationTestUtils; import org.apache.kafka.streams.kstream.ForeachAction; @@ -52,23 +50,16 @@ import java.io.IOException; import java.util.Arrays; import java.util.HashMap; -import java.util.Iterator; import java.util.Map; import java.util.Properties; @Category({IntegrationTest.class}) public class GlobalKTableIntegrationTest { private static final int NUM_BROKERS = 1; - private static final Properties BROKER_CONFIG; - static { - BROKER_CONFIG = new Properties(); - BROKER_CONFIG.put("transaction.state.log.replication.factor", (short) 1); - BROKER_CONFIG.put("transaction.state.log.min.isr", 1); - } @ClassRule public static final EmbeddedKafkaCluster CLUSTER = - new EmbeddedKafkaCluster(NUM_BROKERS, BROKER_CONFIG); + new EmbeddedKafkaCluster(NUM_BROKERS); private static volatile int testNo = 0; private final MockTime mockTime = CLUSTER.time; @@ -229,46 +220,14 @@ public boolean conditionMet() { } }, 30000L, "waiting for final values"); } - - @Test - public void shouldRestoreTransactionalMessages() throws Exception { - produceInitialGlobalTableValues(true); - startStreams(); - - final Map<Long, String> expected = new HashMap<>(); - expected.put(1L, "A"); - expected.put(2L, "B"); - expected.put(3L, "C"); - expected.put(4L, "D"); - - TestUtils.waitForCondition(new TestCondition() { - @Override - public boolean conditionMet() { - ReadOnlyKeyValueStore<Long, String> store = null; - try { - store = kafkaStreams.store(globalStore, QueryableStoreTypes.<Long, String>keyValueStore()); - } catch (InvalidStateStoreException ex) { - return false; - } - Map<Long, String> result = new HashMap<>(); - Iterator<KeyValue<Long, String>> it = store.all(); - while (it.hasNext()) { - KeyValue<Long, String> kv = it.next(); - result.put(kv.key, kv.value); - } - return result.equals(expected); - } - }, 30000L, "waiting for initial values"); - System.out.println("no failed test"); - } - private void createTopics() throws InterruptedException { streamTopic = "stream-" + testNo; globalTableTopic = "globalTable-" + testNo; CLUSTER.createTopics(streamTopic); CLUSTER.createTopic(globalTableTopic, 2, 1); } - private void startStreams() { kafkaStreams = new KafkaStreams(builder.build(), streamsConfiguration); kafkaStreams.start(); @@ -292,29 +251,20 @@ private void produceTopicValues(final String topic) throws Exception { } private void produceInitialGlobalTableValues() throws Exception { - produceInitialGlobalTableValues(false); } - private void produceInitialGlobalTableValues(final boolean enableTransactions) throws Exception { Properties properties = new Properties(); if (enableTransactions) { properties.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, "someid"); properties.put(ProducerConfig.RETRIES_CONFIG, 1); } IntegrationTestUtils.produceKeyValuesSynchronously(globalTableTopic, Arrays.asList(new KeyValue<>(1L, "A"), new KeyValue<>(2L, "B"), new KeyValue<>(3L, "C"), new KeyValue<>(4L, "D")), new KeyValue<>(4L, "D")), TestUtils.producerConfig(CLUSTER.bootstrapServers(), LongSerializer.class, StringSerializer.class, properties), mockTime, enableTransactions); StringSerializer.class), mockTime); } private void produceGlobalTableValues() throws Exception { diff --git a/streams/src/test/java/org/apache/kafka/streams/integration/utils/IntegrationTestUtils.java b/streams/src/test/java/org/apache/kafka/streams/integration/utils/IntegrationTestUtils.java index fe897c7ac30..441549dca77 100644 --- a/streams/src/test/java/org/apache/kafka/streams/integration/utils/IntegrationTestUtils.java +++ b/streams/src/test/java/org/apache/kafka/streams/integration/utils/IntegrationTestUtils.java @@ -179,16 +179,38 @@ public static void purgeLocalStreamsState(final Properties streamsConfiguration) producer.flush(); } } + public static <K, V> void produceAbortedKeyValuesSynchronouslyWithTimestamp(final String topic, final Collection<KeyValue<K, V>> records, final Properties producerConfig, final Long timestamp) throws ExecutionException, InterruptedException { try (final Producer<K, V> producer = new KafkaProducer<>(producerConfig)) { producer.initTransactions(); for (final KeyValue<K, V> record : records) { producer.beginTransaction(); final Future<RecordMetadata> f = producer .send(new ProducerRecord<>(topic, null, timestamp, record.key, record.value)); f.get(); producer.abortTransaction(); } } } public static <V> void produceValuesSynchronously(final String topic, final Collection<V> records, final Properties producerConfig, final Time time) public static <V> void produceValuesSynchronously(final String topic, final Collection<V> records, final Properties producerConfig, final Time time) throws ExecutionException, InterruptedException { IntegrationTestUtils.produceValuesSynchronously(topic, records, producerConfig, time, false); } public static <V> void produceValuesSynchronously(final String topic, final Collection<V> records, final Properties producerConfig, final Time time, final boolean enableTransactions) throws ExecutionException, InterruptedException { public static <V> void produceValuesSynchronously(final String topic, final Collection<V> records, final Properties producerConfig, final Time time, final boolean enableTransactions) throws ExecutionException,

InterruptedException { final Collection<KeyValue<Object, V>> keyedRecords = new ArrayList<>(); for (final V value : records) { final KeyValue<Object, V> kv = new KeyValue<>(null, value); @@ -240,10 +262,9 @@ public static void waitForCompletion(final KafkaStreams streams, public static <K, V> List<KeyValue<K, V>> waitUntilMinKeyValueRecordsReceived(final Properties consumerConfig, final String topic, final int expectedNumRecords) throws InterruptedException { - return waitUntilMinKeyValueRecordsReceived(consumerConfig, topic, expectedNumRecords, DEFAULT_TIMEOUT); } - + /** * Wait until enough data (key-value records) has been consumed. * ---------- ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org