Item 230
git_comments:

1. Producer will assign
2. replace "deleteme" with a tombstone
3. the buffer should serialize the buffer time and the value as byte[], which we can't compare for equality using ProducerRecord. As a workaround, I'm deserializing them and shoving them in a KeyValue, just for ease of testing.
4. flush the buffer into a list in buffer order so we can make assertions about the contents.
5. flush everything to the changelog
6. * 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.
7. Several things to note: * The buffered records are ordered according to their buffer time (serialized in the value of the changelog) * The record timestamps are properly restored, and not conflated with the record's buffer time. * The keys and values are properly restored * The record topic is set to the changelog topic. This was an oversight in the original implementation, which is fixed in changelog format v1. But upgraded applications still need to be able to handle the original format.
8. Several things to note: * The buffered records are ordered according to their buffer time (serialized in the value of the changelog) * The record timestamps are properly restored, and not conflated with the record's buffer time. * The keys and values are properly restored * The record topic is set to the original input topic, *not* the changelog topic * The record offset preserves the origininal input record's offset, *not* the offset of the changelog record
9. nothing to do.
10. As we add more buffer implementations/configurations, we can add them here
11. expected
12. sizes of key and value
13. value.context.timestamp value.context.offset size of topic
14. partition number of headers
15. not handling the null topic condition, because we believe the topic will never be null when we serialize
16. not handling the null condition because we believe topic will never be null in cases where we serialize
17. create multiple partitions as a trap, in case the buffer doesn't properly set the partition on the records, but instead relies on the default key partitioner
18. expect all post-suppress records to keep the right input topic
19. note, we send all input records to partition 0 to make sure that supppress doesn't erroneously send records to other partitions.

git_commits:

1. **summary:** KAFKA-7895: fix Suppress changelog restore (#6536)
   message: KAFKA-7895: fix Suppress changelog restore (#6536) Several issues have come to light since the 2.2.0 release: upon restore, suppress incorrectly set the record metadata using the changelog record, instead of preserving the original metadata restoring a tombstone incorrectly didn't update the buffer size and min-timestamp Reviewers: Guozhang Wang <wangguoz@gmail.com>, Matthias J. Sax <mjsax@apache.org>, Bruno Cadonna <bruno@confluent.io>, Bill Bejeck <bbejeck@gmail.com>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** KAFKA-7895: fix Suppress changelog restore
   body: Several issues have come to light since the 2.2.0 release: * upon restore, `suppress` incorrectly set the record metadata using the changelog record, instead of preserving the original metadata * restoring a tombstone incorrectly didn't update the buffer size and min-timestamp ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)

github_pulls_comments:

1. Hi @ableegoldman @cadonna @guozhangwang , Please review these bugfixes for Suppression when you get the chance. The reporter has confirmed it fixes the duplicates in his repro as well: https://issues.apache.org/jira/browse/KAFKA-7895?focusedCommentId=16810770&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-16810770 I'm still chasing down a different bug to do with the interaction between Suppression and EOS, but I don't think it's related and should be a separate PR. Thanks, -John
2. Thanks @vvcephei for the PR. Do you want me to review it now or wait for some further investigations?
3. Sorry for the delay in responding, @guozhangwang . I think we should review this now. Right now, it seems like the EOS thing is a different mechanism, so I plan to do a separate PR once I figure it out.
4. Thanks for the review, @bbejeck . I've addressed your comments and added an integration test that checks that the topic name is right, at least. It seems a little tricky to deterministically verify the incoming event's offset or partition number.
5. To actually generate duplicates under the condition I fixed, we'd have to run the application for a pretty long time.
6. I've only made a brief pass on the bullet points that's listed on the description and these fixes lgtm. cc @mjsax @ableegoldman to make another thorough pass over it.
7. Thanks for the review, @cadonna ! I'm currently seeing if I can get the integration test to spit out duplicates. I'll address your comments in the next commit.
8. Ok @bbejeck and @cadonna , I managed to update the integration test to re-create the duplication bug on trunk (you can see this in action if you check out https://github.com/vvcephei/kafka/tree/testing-test , which has this test cherry-picked onto trunk). I've verified that it fails on trunk both because the topic is wrong on suppressed records *and* because it produces duplicates. Thanks for inducing me to go the extra mile, @bbejeck . I think this is ready for final reviews. -John
9. Flaky test failure: https://issues.apache.org/jira/browse/KAFKA-7965?focusedCommentId=16815432&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-16815432 Retest this, please.
10. SUCCESS 10841 tests run, 69 skipped, 0 failed. --none--
11. Thanks for the reviews, @bbejeck, @cadonna, and @mjsax . I've addressed all comments. Do you mind taking another look?
12. Ok, in adopting the PR comments regarding assuming the topic will never be null, I broke a whole bunch of tests. I've fixed it with the following strategy: * the topic may be null during size estimation * the topic will never be null when serializing the record context
13. **body:** Java 8 failed with flaky ConsumerBounceTest. This test was purportedly fixed after my build started, so I'm giving it the benefit of the doubt and not reporting it (https://issues.apache.org/jira/browse/KAFKA-7965). Retest this, please.
    label: test
14. Hey, the tests passed! Do you think we're ready to merge this, @mjsax ?
15. @vvcephei I've made another pass LGTM. Just ping us when the last bits are addressed and we can merge
16. Thanks, all. @guozhangwang and @mjsax , I believe the only remaining unresolved thread is regarding the header check above. Thanks!

17. Ok, @guozhangwang and @mjsax , I think I've addressed your feedbacks. I also did a sweep for any more missed feedback (sorry about that). Please take another look when you get the chance (also @bbejeck and @cadonna). Thanks for the reviews!
18. **body:** Java11 failed. Flaky tests tracked in Jira. Java8 passed. Retest this please.
    label: test
19. Merged #6536 into trunk.
20. Thanks for the fix @vvcephei!

github_pulls_reviews:

1. Why this change from `long` to `int`?
2. Is there a chance here that the cumulative size of bytes for all headers and values exceeds max int? I realize the probability is very low, but should we have a check here and throw an exception in case of an overflow (maybe I'm overthinking this?)
3. why not use `Objects.requireNonNull` here and in the next branch?
4. **body:** nit: args on separate line
   label: code-design
5. **body:** nit: args on separate line
   label: code-design
6. If we are serializing the `ContextualRecord` could we ever hit this case?
7. Unless I'm missing something aren't we restoring the record with the changelog `offset`, `topic` etc?
8. **body:** Could you please assign `Integer.BYTES` to a variable with a meaningful name and use that variable here? It would make the code bit clearer.
   label: code-design
9. **body:** `long` is an unnecessarily large data type for `size`, since arrays can only be indexed with integers (thus, a byte array cannot hold more than a "max int" number of bytes).
   label: code-design
10. **body:** That's a good idea. I agree it's unlikely, and if it does happen, we'd wind up with an exception while serializing later, but we could detect it sooner here and throw a nicer error.
    label: code-design
11. Only because `requireNonNull` throws a `NullPointerException`, which I felt was less appropriate than an `IllegalArgumentException` in this case.
12. oof! Sorry, I missed these long lines.
13. Yep, the currently released versions of Suppress are already serializing changelog messages without this header. The purpose of this check is to be able to transition to new Streams versions in a backward-compatible way.
14. Yeah, this is the (incorrect) behavior of the current code. Luckily, this will only happen for the very small window of emitting records that are currently buffered when you upgrade Streams to "this" version. Once those are flushed out, the buffer will always be sending/receiving "v1" changelog messages that correctly encode the upstream topic/offset. I figured the best way to handle this transition is to just preserve the existing behavior for "old format" messages, rather than try to do something clever.
15. **body:** Also here, could you please give `Long.BYTES` a meaningful name?
    label: code-design
16. I doubt that we could exceed it. Note, that `max.message.byte` is an integer broker side config, and I believe it applies to the serialized version of a message. \cc @guozhangwang @hachikuji Is this correct? Of course, a user could emit a new key-value pairs with huge key/value/header arrays, however, this message could not be stored in Kafka itself either. If we want a guard, we should use `long size` internally here, and check if it exceed `INT_MAXVALUE` and throw of course.
17. This does not guard against "double overflow". I think, if we apply a check, we should check for all corner cases or not check at all.
18. **body:** nit: `headerKey` -> `headerKeyBytes` (same for `headerValues` below)
    label: code-design
19. How could `topic` be `null`?
20. `valueLength` could be `-1`
21. Shouldn't headers be `null` or contain exactly one header with `key=="v"` and non-null value? If, the second part of "or" should never be true?
22. **body:** nit: move `key` to new line
    label: code-design
23. I cannot follow. What is this discussion about?
24. Why do we need this check?
25. **body:** The argument that `NullPointerException` is not appropriate holds for almost all cases when `requireNonNull` in use. Thus, I would prefer code consistency and use `requireNonNull` here, too.
    label: code-design
26. Why this change?
27. Why change from `v` to `k` ?
28. **body:** nit: `produceSynchronouslyToPartition0` -> `produceSynchronouslyToPartitionZero`
    label: code-design
29. **body:** Is there a way to check against wrapping all the way around to positive again? Even a long could wrap around in principle... This is starting to get a little complicated. Maybe we should just fall back on the jvm to catch us when we try to make and populate the byte array after all.
    label: code-design
30. **body:** I could see making an assumption that a specific ConsumerRecord is always has a topic (because that's where they come from), but this is the ProcessorRecordContext. The class has no null checks on the topic field, so rather than tracing all the usages to find out whether it can ever be null, in main code or test code, it's simpler and bulletproof just to check for it.
    label: code-design
31. good call!
32. If it's an "old format" changelog record, the headers would have been copied from the record context, in which case they could be null or non-null, but still not contain a "v". Of course, it could contain a "v" which has nothing to do with our algorithm here, but I think that situation is undetectable. And unlikely, as well.
33. **body:** > Is there a way to check against wrapping all the way around to positive again? I think you would need to check after each modification... This would of course become costly. > Even a long could wrap around in principle... True. But I think the risk would be minimal in practice for this case. > Maybe we should just fall back on the jvm to catch us when we try to make and populate the byte array after all. Not sure what you mean by this?
    label: code-design
34. `ProcessorRecordContext` would only set the topic to `null` for calls into `init()`, `close()`, and punctuation callbacks. Thus, it should never be `null` here IMHO and I believe it would indicate a bug, if it's `null` here. Hence, this check might masked a bug and we should rather fail with a NPE for this case. \cc @guozhangwang to confirm.
35. I don't think so. We never write headers in the changelogger. Note, that the changelog topic is used to recover the store content. However, rows in a store only have a key and a value. There is no header that we could write, because the on put, the current record header does not related to the store content. Similarly, `suppress()` serializes the whole record context and store it in the value IIRC.
36. **body:** I meant that maybe we should just remove the check.
    label: code-design
37. We're setting the record metadata incorrectly here, which is actually the bug I'm fixing with the new format. It's just that there's apparently nothing better to do that continue handling the old changelog format the same way, since the format itself is missing the information we need.
38. good call. So, we're covered. Thanks for this analysis.

39. **body:** Thanks for the information. I think this is getting a little complicated, and we'll just drop the check. If the value is too big, we won't be able to serialize it later anyway.
    label: code-design
40. This is one of the bugs that this PR fixes. The correct operation of this class depends on maintaining `minTimestamp` on every mutation. If a future code change breaks that contract, it would re-introduce a variant of our bug, and it would be just as subtle as it was this time. This check safeguards against that eventuality, by checking on our invariant at the spot we depend on it.
41. **body:** When I wrote the tests the first time around, I overlooked a simpler application to test suppression. Switching to this simpler application made it possible to directly test for the bug (duplicated suppress output).
    label: code-design
42. Instead of grouping on the values anymore, now I'm just counting events. I just didn't realize that simpler aggregation was available when I wrote the tests the first time.
43. > or not check at all. Fine with we to remove the check.
44. Just reviewing this PR, it reminds me on this JIRA: https://issues.apache.org/jira/browse/KAFKA-8235 Seems `firstKey` would throw if `sortedMap` is empty.
45. **body:** Well, my point is, that the check can be simplified. I don't think that `record.headers() == null` can be true; it's guaranteed that there is a headers object. Not sure if we can simplify the second check. It iterators over the headers map and does String comparison to find a header with key `v` -- seems to be rather heavy.
    label: code-design
46. Ack. So we don't need it, but we want it :)
47. Oof! You're absolutely right. What a strange api asymmetry...
48. Ok, so I have a clarification: this method is used to estimate the size of the record context in memory, for a variety of usages not restricted to processing. Including some of the cases where you point out it may be null, so we do need the null-check in this method. In the de//serialize methods below, I've removed the null-check, following the spirit of your feedback.
49. This seems not fixed yet?
50. If I understand correct, a record read from the changelog topic should only be: 1) having no headers at all (old version) 2) having a singleton header with `v -> byte`. All other cases should indicate a bug in the code. So it seems we can just check if `record.headers() == null`, and inside the if condition though, we should also check the `v` and assume it's always there (if not then throw illegal state), and switching on the byte value: 1) byte == 1: do as below. 2) otherwise: to not support forward compatibility, we can just throw unsupported.
51. **body:** We were previously setting the changelog record headers from the record headers. I really don't think it's safe to just assume that record headers are always empty for records inside of Streams. Also, I'd like to point out that @guozhangwang thinks that old-format headers are _always_ null, and @mjsax think that they are _never_ null, but _always_ empty... The point is, header handling is partially undefined inside Streams. I'd be very much in favor of being defensive in the face of such ambiguity, rather than making potentially faulty assumptions, since the risk is a runtime exception or even data corruption (if we mis-interpret the binary format, but it happens to de-serialize without exceptions into gibberish). Further, none of this feedback is pointing out a problem with the code, just that there might be one unnecessary clause in the boolean expression (which, I still am not convinced is unnecessary after all).
    label: code-design
52. Gah. Thanks for the catch!
53. Thanks for the explanation @vvcephei , makes sense. One thing I'd still like to ask though: from the current code it seems we do not check the byte of `v` if it indeed exist, while I think to be safer we should still check it, and throw exception indicating that we do not support forward compatibility when the format has evolved. WDYT?
54. This comment is acked but not addressed. Did it slip? Or did you decide to ignore it?
55. **body:** I understand your argument. However, I am not 100% sure if this does not introduce a weird asymmetry? Also, we only estimate the size _if_ we serialize the context. We don't estimate in-memory/object space usage, do we?
    label: code-design
56. Why this block?
57. - It's a contract that `KafkaConsumers` _guarantees_ that `header != null`. \cc @hachikuji to confirm. - And we know that KafkaStreams never writes headers into changelog topics. Thus, I don't see any reason to check for something that we know is the case, ie, we know that `header.size() == 0` in old format. For everything else, we could throw `IllegalStateException`. Of course the header API is limiting and we cannot get `size()` and thus `record.headers().lastHeader("v") == null)` is what we need to do... :(-- but we can safely remove the first `null` check -- it could even mask a bug and we should rather fail for this case. We can also do a version check as suggested by Guozhang.
58. > we know that KafkaStreams never writes headers into changelog topics. We're restoring records that we wrote like this: https://github.com/apache/kafka/blob/trunk/streams/src/main/java/org/apache/kafka/streams/state/internals/InMemoryTimeOrderedKeyValueBuffer.java#L240 We write headers to the suppression changelog topic. Sorry if that wasn't clear, that's why I'm insisting we have to check it. If there's a contract that headers are never null, then I agree we can remove that check. In that case, we should add null-checks to the ConsumerRecord constructor.
59. Yes, we do. This is also used for computing the size of the record cache to determine cached state store flushing, etc. (I was a little surprised when a bunch of "unrelated" tests failed as a result of the change I tried here). I shared your concern about the asymmetry, but I think it's actually ok, even if you only consider the suppression buffer. The `sizeBytes` method is an estimation of the resident size of this object in memory, which is only loosely related to the size of the serialized form of the data in this object.
60. Oy, it slipped. Fixing it now.
61. I shouldn't have acked it until I fixed it; that's why I overlooked it on later passes.
62. **body:** Actually, the latter conversation with @mjsax about the relationship (or lack of one) between this and the serialized form makes me think maybe this response was incorrect. I'll move it back to long.
    label: code-design
63. **body:** Ok, I've had a chat with @hachikuji , and he confirmed that the headers _should_ never be null. I did a quick audit of code base as well, and agree that this should be the case. I added a null-check to the header and docs to this effect in f5bd0999c (in this PR), and then removed the null-check. I also added an exception if the record has a "v" header, but it isn't `1`. I'm wondering, @guozhangwang , if we should just assume in this case that the record is old-format, and happened to already have a "v" header. The alternative thing is you could have run Streams with a _newer_ version, and then downgraded. In that case, it would be wrong to try restoring as old-format, and an exception would be preferable. I'm leaning toward the exception, since people haven't been using Suppress that long, so the risk of a spurious "v" header is low. WDYT?
    label: code-design
64. **body:** It just prevents polluting the method scope with the temp variable `topicBytes`
    label: code-design
65. > We're restoring records that we wrote like this: https://github.com/apache/kafka/blob/trunk/streams/src/main/java/org/apache/kafka/streams/state/internals/InMemoryTimeOrderedKeyValueBuffer.java#L240 > > We write headers to the suppression changelog topic. Sorry if that wasn't clear, that's why I'm insisting we have to check it. Arg. We should not have done this... But serialize the header in KS and add them to the value. Well, too late now I guess. Or could we switch the format? Note sure what the impact on the upgrade path would be? Thought? > I added a null-check to the header and docs to this effect in f5bd099 (in this PR), and then removed the null-check. +100
66. Ack. Thanks for being patient with me :)

jira_issues:

1. **summary:** Ktable supress operator emitting more than one record for the same key per window
   description: Hi, We are using kstreams to get the aggregated counts per vendor(key) within a specified window. Here's how we configured the suppress operator to emit one final record per key/window. {code:java} KTable<Windowed<Integer>, Long> windowedCount = groupedStream .windowedBy(TimeWindows.of(Duration.ofMinutes(1)).grace(ofMillis(5L))) .count(Materialized.with(Serdes.Integer(),Serdes.Long()))

.suppress(Suppressed.untilWindowCloses(unbounded())); {code} But we are getting more than one record for the same key/window as shown below. {code:java} [KTABLE-TOSTREAM-0000000010]: [131@1549067040000/1549067100000], 1039 [KTABLE-TOSTREAM-0000000010]: [131@1549067040000/1549067100000], 1162 [KTABLE-TOSTREAM-0000000010]: [9@1549067040000/1549067100000], 6584 [KTABLE-TOSTREAM-0000000010]: [88@1549067040000/1549067100000], 107 [KTABLE-TOSTREAM-0000000010]: [108@1549067040000/1549067100000], 315 [KTABLE-TOSTREAM-0000000010]: [119@1549067040000/1549067100000], 119 [KTABLE-TOSTREAM-0000000010]: [154@1549067040000/1549067100000], 746 [KTABLE-TOSTREAM-0000000010]: [154@1549067040000/1549067100000], 809{code} Could you please take a look? Thanks Added by John: Acceptance Criteria: * add suppress to system tests, such that it's exercised with crash/shutdown recovery, rebalance, etc. ** [https://github.com/apache/kafka/pull/6278] * make sure that there's some system test coverage with caching disabled. ** Follow-on ticket: https://issues.apache.org/jira/browse/KAFKA-7943 * test with tighter time bounds with windows of say 30 seconds and use system time without adding any extra time for verification ** Follow-on ticket: https://issues.apache.org/jira/browse/KAFKA-7944

jira_issues_comments:

1. [~vvcephei] Not sure how quickly we can figure this out. However, 2.1.1 is not released yet and there might be another RC... For this case, if we are quickly enough, we could get it into 2.1.1. Don't think it's a blocker for 2.1.1 though. \cc [~cmccabe]
2. I am experiencing what appears similar/related. I’m seeing quite a large number of duplicates being created when my topology comes back up after being restarted. (Caching is disabled and Im using a short commit internal). I’m using a groupBy key -> windowBy (time) -> aggregate -> suppress. The suppress is configured until window closes, unbounded, window-size 1 hour, grace 1 min. Its there are a smaller number of duplicates created whilst the topology is running, but its much smaller compared to when its restarted.
3. Hi [~mjsax], Thanks for thinking about that. I've been hacking on this issue for a little over a day now, with no luck so far. Given that we don't know how long it will take to repro the issue or figure it out and fix it, I'd hold off on blocking the release. If I have a breakthrough, I'll figure out what stage the release is in. It is a serious report, though, so in either case, once we have a fix, I would request an immediate bugfix release. Thanks, -John (cc [~cmccabe])
4. Quick update: I have added a suppression operator to Streams's system tests, and I believe that I have a local repro of the bug. I'll dig in more tomorrow and update everyone with status as I learn more. Thanks, -John
5. Great! Thank you John for the update.
6. I think I figured it out. The suppression operator depends on a guarantee from upstream windowed aggregations that they will not send any further events for a window once its grace period expires. The windowed aggregations themselves correctly implement this guarantee, but record caches can violate it, since they'll hold onto events until the next commit/flush. This suggests that a workaround would be to disable caching in the windowed aggregation prior to suppression, but I didn't try it out. Depending on the specific topology, this might not be sufficient. It just so happens that in all my tests, I'd disabled caching, probably with the idea of making the output deterministic. I didn't predict that it would affect correctness. I have a fix for it, by changing the way that stream-time is accounted for. I'm running the system tests now to be sure the fix is ok. Once a preliminary PR ([https://github.com/apache/kafka/pull/6231)] is merged and the system tests pass, I'll clean up my branch and send a PR. Once I submit the PR, maybe some intrepid folks out there can pull my branch and try it out. Does this explanation make sense? Thanks, -John
7. Hi John, thanks for looking into it. One thing though is that [~mbragg] reported he sees similar issues in his environment while he's already disabled caching, does that mean there are other issues not related to caching layer as well?
8. Thanks for the reminder, [~guozhang]. I think that the same mechanism could cause duplicates as well during recovery, since stream time is currently determined by the input partitions, but the recovery is from the changelog topic, so recovered events may all appear to be arriving at the same stream time. If I'm right about this, then the fix I have in mind should address that as well. However, it is troubling that Michael also said that he observed duplicates during steady-state processing, post recovery, with caching disabled. There may be something else going on as well. The system tests already exercise crash and shutdown recovery, so to complete the picture, I'll make sure I add a system test with caching disabled (if there's not already one).
9. vvcephei commented on pull request #6278: KAFKA-7895: Fix stream-time reckoning for suppress URL: https://github.com/apache/kafka/pull/6278 * Add suppress to system tests * Move stream-time reckoning from Task into Processor Even within a Task, different Processors have different perceptions of time, due to record caching on stores and in suppression itself, and in general, due to any processor logic that may hold onto records arbitrarily and emit them later. Thanks to this, we can't rely on the whole task existing in the same "instant" of stream-time. The solution is for each processor node that cares about stream-time to track it independently. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- 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
10. Hi all, I'm proposing [https://github.com/apache/kafka/pull/6278] as a fix to this issue. If anyone is willing to build my branch and verify if it fixes the issue for them, I would be grateful. Your code reviews are also appreciated. Thanks, -John
11. https://github.com/apache/kafka/pull/6278
12. Hi [~vvcephei] Thanks for your proposed fix, sounds promising! To clarify my initial comment, I meant to say that its _possible_ there's duplicates during steady processing... but if so the number would be much less then during restore of the state stores; where there is a massive spike. Sorry for the confusion.
13. mjsax commented on pull request #6278: KAFKA-7895: Fix stream-time reckoning for suppress URL: https://github.com/apache/kafka/pull/6278 ---------- 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
14. vvcephei commented on pull request #6286: KAFKA-7895: fix stream-time reckoning for Suppress URL: https://github.com/apache/kafka/pull/6286 * Add suppress to system tests * Move stream-time reckoning from Task into Processor Even within a Task, different Processors have different perceptions of time, due to record caching on stores and in suppression itself, and in general, due to any processor logic that may hold onto records arbitrarily and emit them later. Thanks to this, we can't rely on the whole task existing in the same "instant" of stream-time. The solution is for each processor node that cares about stream-time to track it independently. See also #6278 ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- 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
15. Hey [~vvcephei] What would be a typical ETA of this fix being available in a release? Are there any pre-releases published anywhere? Sorry to be a pain! Thanks
16. bbejeck commented on pull request #6286: KAFKA-7895: fix stream-time reckoning for Suppress (2.2) URL: https://github.com/apache/kafka/pull/6286 ---------- 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
17. vvcephei commented on pull request #6325: KAFKA-7895: fix stream-time reckoning for Suppress (2.2) (#6286) URL: https://github.com/apache/kafka/pull/6325 Even within a Task, different Processors have different perceptions of time, due to record caching on stores and in suppression itself, and in general, due to any processor logic that may hold onto records arbitrarily and emit them later. Thanks to this, we can't rely on the whole task existing in the same "instant" of stream-time. The solution is for each processor node that cares about stream-time to track it independently. On the side, backporting some internally-facing code maintainability updates ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- 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
18. Hi, [~mbragg], Sorry for the delay. Thanks for the clarification. I'm very curious if the patch resolves the problem for you. The 2.2 release should be very soon. They are currently voting on the release candidate in the dev mailing list. Actually, the release candidate message contains information about testing the release candidate: [https://lists.apache.org/list.html?dev@kafka.apache.org:lte=1M:%5BVOTE%5D%202.2.0%20RC1] Please let me know if you have any trouble with it. Thanks, -John
19. bbejeck commented on pull request #6325: KAFKA-7895: fix stream-time reckoning for Suppress (2.1) (#6286) URL: https://github.com/apache/kafka/pull/6325 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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

20. Hi I am still seeing this behaviour using the 2.2.0 Maven artifacts from [https://repository.apache.org/content/groups/staging] With the topology below, I invariably get emissions of previously emitted windows for the same key on restart of my streams application, and sometimes the re-emitted windows have earlier timestamps and aggregated data that is consistent with the latter part of the window being lost (i.e., state seems to be resetting to an earlier version): {code:java} return sourceStream .groupByKey() .windowedBy(TimeWindows.of(rollupInterval).grace(config.getGracePeriodDuration())) .aggregate(() -> rollupFactory.createRollup(windowDuration), aggregator, Materialized.<String, R, WindowStore<Bytes, byte[]>>as(outputTopic + "_state") .withValueSerde(rollupSerde)) .suppress(Suppressed.untilWindowCloses(BufferConfig.unbounded()) .withName(outputTopic + "_suppress_state")) .toStream((stringWindowed, rollup) -> stringWindowed.key()); {code} I have tried disabling caching using: {code:java} streamsConfiguration.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0); {code} but this does not appear to make a difference. There also seems to be no difference in behaviour between 2.2.0 and 2.1.1. Regards Andrew
21. Hi [~AndrewRK], Sorry to hear that! Looking at your topology, I'd expect it to work as advertised (obviously), and it also looks very similar to what I've tested heavily, so I might need to get some more help from you to investigate the cause. I have a couple of follow-up questions... # Is this restarting after a clean shutdown or a crash? (i.e., how do you stop and restart the app?) # Do you have EOS enabled? (If you don't, can you try to repro with EOS on?) Thanks, -John
22. **body:** Hi John Roesler To answer your first question, I have a shutdown hook in place that seems to be working correctly according to the slf4j logs (the application changes state from RUNNING -> PENDING_SHUTDOWN -> NOT_RUNNING). {code:java} Runtime.getRuntime().addShutdownHook(new Thread(streams::close)); {code} I do have exactly-once configured. My settings are as follows (commit interval of 10s): {code:java} streamsConfiguration.put(StreamsConfig.APPLICATION_ID_CONFIG, appConfig.getApplicationID()); streamsConfiguration.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, appConfig.getBootstrapServerList().stream().map(HostPort::toString).collect(Collectors.joining(","))); streamsConfiguration.put(StreamsConfig.STATE_DIR_CONFIG, appConfig.getStateDirectory()); streamsConfiguration.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, appConfig.getCommitIntervalDuration().toMillis()); streamsConfiguration.put(StreamsConfig.DEFAULT_TIMESTAMP_EXTRACTOR_CLASS_CONFIG, com.booking.infra.rollup.kafka.TimestampedValueTimestampExtractor.class); streamsConfiguration.put(StreamsConfig.PROCESSING_GUARANTEE_CONFIG, StreamsConfig.EXACTLY_ONCE); {code} When I test I start with a new docker-based Kafka cluster and delete the application state directory. I feed the app with enough input to generate one or two commits and then hit Ctrl-C, which appears to shut everything down correctly. I then restart the app leaving the state directory untouched. My code does not call streams.cleanup(). Almost without fail this will produce repeated entries for the same key/window. Sometimes the repeated entries will be different, in which case the repeat matches an older entry in the source KTable. I do not have to stress the app in any way to cause this, which makes me wonder if I am doing something fundamentally wrong. Here is an example of the repeated output: {code} 0.0026: 1553591980510: monitors.group1:stat.stat1: time=1553591980510 wstart=1553591980000 wsize=1000 count=12 sum=1044 sumSq=91400 min=76 max=98 last=98 0.0041: 1553591980057: monitors.group1:stat.stat1: time=1553591980057 wstart=1553591980000 wsize=1000 count=2 sum=154 sumSq=11860 min=76 max=78 last=78 {code} And here is are the corresponding entries in the KTable changelog: {code} 0.0026: 1553591980057: [monitors.group1:stat.stat1@1553591980000/1553591981000]: time=1553591980057 wstart=1553591980000 wsize=1000 count=2 sum=154 sumSq=11860 min=76 max=78 last=78 0.0028: 1553591987079: [monitors.group1:stat.stat1@1553591980000/1553591981000]: time=1553591980510 wstart=1553591980000 wsize=1000 count=12 sum=1044 sumSq=91400 min=76 max=98 last=98 {code} Please let me know if you need any more information, or if you need me to run any more tests. Regards Andrew
    label: code-design
23. Thanks for the extra details, [~AndrewRK], I'll try setting up a scenario similar to what you've described and see if I get the same behavior. I doubt this is the problem, but just to check all the boxes, when you read the output topic, have you set the consumer to "read_committed"? -John
24. Hi John Roesler Yes, my consumer's isolation level is set to read_committed. Regards Andrew
25. EDIT: actually, nevermind. After replicating your application, I was able to reproduce what you're seeing. I'll let you know when I figure out what's going on. Thanks! Also, can you elaborate on how `TimestampedValueTimestampExtractor` behaves? Thanks, -John
26. Here's the repro I put together. It'll take me a little while to debug it, but I wanted to share my approach. [https://github.com/vvcephei/suppress-demo] -John
27. **body:** Quick update: The main source of the duplicates seems to be this odd situation where the suppression buffer is sometimes sending its records to the wrong changelog partition. This results in duplicates because those changelog partitions are handled independently, so the message sent to the wrong partition will be emitted in addition to the ones sent to the right partition. It's still not clear why some records are being logged to the wrong partition. When I fully understand why this is happening, I should also be able to explain why none of the existing tests have caught this condition, when it's so easy to reproduce with your application.
    label: code-design
28. Status update: I found a couple of subtle bugs in the logic for restoring tombstones from the changelog for the suppression buffer. Running with my patched version of Streams, I was able to restart the application in my suppress-demo six times in a row without seeing any duplicates. (Previously, I'd see it on the first restart). So, I feel pretty good about this fix. I'm cleaning up my branch and will have a PR for review by tomorrow morning. I think the reason that none of the existing integration/system tests caught it is just that they don't run over a long enough time scale. I'm working on regression tests for these specific bugs, which I'll include as a part of my PR. I'll create a ticket to add a more realistic system test to exercise Streams better and expose bugs like this.
29. vvcephei commented on pull request #6536: KAFKA-7895: fix Suppress changelog restore URL: https://github.com/apache/kafka/pull/6536 Several issues have come to light since the 2.2.0 release: * upon restore, `suppress` incorrectly set the record metadata using the changelog record, instead of preserving the original metadata * restoring a tombstone incorrectly didn't update the buffer size and min-timestamp ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
30. Hey [~AndrewRK], I've created a PR to fix the issues I've found from your report. Do you mind testing my patch and confirming that it fixes your issues? [https://github.com/apache/kafka/pull/6536] Note: I'm currently seeing no duplicates if I close Streams gracefully (using SIGTERM + shutdown hook), but I am still seeing duplicates if I terminate Streams abruptly (using SIGKILL). I think this has a separate root cause, and it wasn't part of your report, so I'm submitting this PR now, while I continue to debug the further issues. Thanks, -John
31. Hi John Roesler I can confirm that I am no longer seeing duplicates. Thanks a lot for your help. Regards Andrew
32. No problem. Thanks for _your_ help! The PR is being reviewed now. Once it's merged, I'll request it to be cherry-picked to all versions that have Suppress, and then request bugfix releases for them. As an update, I've tracked down the problem where I'm still seeing duplicates when I crash the application, even with EOS enabled. The problem is that Streams flushes the state stores in no particular order. If there is a cached state store upstream of the suppression, and Streams flushes the suppression buffer first, and then the cached state store second, the cached data will get flushed and processed by Suppress, and there may be processing results that Suppress emits, but the changelog for the buffer won't be updated again. The solution I have in mind is to flush the stores in topological order. In the mean time, disabling caching should eliminate this particular source of duplicates, if someone is looking for a workaround.
33. I'm tracking the incorrect flushing behavior in a separate ticket (KAFKA-8204). This is the root cause of the continued duplicate results, even when EOS is enabled, when the Streams process undergoes an abrubt stop.
34. bbejeck commented on pull request #6536: KAFKA-7895: fix Suppress changelog restore URL: https://github.com/apache/kafka/pull/6536 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
35. Hi [~vvcephei], when do we have a new release for this?
36. vvcephei commented on pull request #6615: KAFKA-7895: fix Suppress changelog restore (#6536) URL: https://github.com/apache/kafka/pull/6615 Several issues have come to light since the 2.2.0 release: upon restore, suppress incorrectly set the record metadata using the changelog record, instead of preserving the original metadata restoring a tombstone incorrectly didn't update the buffer size and min-timestamp Cherry-picked from #6536 / 6538e9e4d6c1f64fe3045a5c3fbfe306277a1bee Reviewers: Guozhang Wang <wangguoz@gmail.com>, Matthias J. Sax <mjsax@apache.org>, Bruno Cadonna <bruno@confluent.io>, Bill Bejeck <bbejeck@gmail.com> ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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

37. vvcephei commented on pull request #6616: KAFKA-7895: fix Suppress changelog restore (#6536) URL: https://github.com/apache/kafka/pull/6616 Several issues have come to light since the 2.2.0 release: upon restore, suppress incorrectly set the record metadata using the changelog record, instead of preserving the original metadata restoring a tombstone incorrectly didn't update the buffer size and min-timestamp Cherry-picked from #6536 / 6538e9e Reviewers: Guozhang Wang <wangguoz@gmail.com>, Matthias J. Sax <mjsax@apache.org>, Bruno Cadonna <bruno@confluent.io>, Bill Bejeck <bbejeck@gmail.com> ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
38. Hi [~songkun], No, this fix is just now merged, and we're working on merging the 2.2 and 2.1 backports. We also need to merge https://issues.apache.org/jira/browse/KAFKA-8204 before we do buxfix releases. I'll send a note to this thread when we do the releases, so all the subscribers can verify they are fixed.
39. [~songkun]: planned release dates are published in the wiki: [https://cwiki.apache.org/confluence/display/KAFKA/Future+release+plan]
40. bbejeck commented on pull request #6615: KAFKA-7895: fix Suppress changelog restore (#6536) URL: https://github.com/apache/kafka/pull/6615 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
41. bbejeck commented on pull request #6616: KAFKA-7895: fix Suppress changelog restore (#6536) URL: https://github.com/apache/kafka/pull/6616 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
42. Update: this bugfix has been merged to trunk (2.3), 2.2, and 2.1 branches. I'm waiting on other bugfixes before requesting patch releases.
43. Thank you [~mjsax] for your kind suggesstion, and [~vvcephei] , thank your timely notification, looking forward the patch release!
44. 2.2.1 release was proposed today. Cf: [https://cwiki.apache.org/confluence/display/KAFKA/Release+Plan+2.2.1]
45. Hi [~vvcephei], [~mjsax], any progress on that?
46. Hi [~songkun], The release process takes a little while, as they have to wait on critical bugfixes to be merged, then build release candidates, and then have a vote for approval. I'd just keep an eye on the release plan page for current status, and if you want to watch it play out "in real time", you can subscribe to the Kafka dev mailing list. There's a thread for the release. Thanks! And I look forward to getting this fix out for you. Thanks again for the report. -John
47. FYI, 2.2.1 RC0 vote is in progress. (https://lists.apache.org/thread.html/3486798e63ae666fc336ce9009f07c7fdf66a96badc1fed63bcbd2ed@%3Cdev.kafka.apache.org%3E) Please feel free to test it out, and reply on the vote thread if you have some trouble with it.
48. Hi [~vvcephei], Great work! I want to test it in my env, but I can't figure how to find this version, since it's still not released, could you please give me some tips where I can download it, thanks! Edit: I just find it now :)
49. As pointed out in the email John linked to, the RC artifact can be downloaded: [https://home.apache.org/~vahid/kafka-2.2.1-rc0/]
50. vvcephei commented on pull request #7373: KAFKA-7895: Revert suppress changelog bugfix for 2.1 URL: https://github.com/apache/kafka/pull/7373 The bugfix from (#6536) (#6616) breaks compatibility with brokers using log format less than 0.11. (Because record headers are not supported in 0.10 brokers) Even though this fix is useful, we should not break broker compatibility in a bugfix release, so I'm reverting just the changelog format change. Note: this re-introduces the suppress bug related to changelog restoration, so folks using suppress are recommended to upgrade to 2.2.1 at least. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes) ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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
51. This won't fix for 2.1.2 due to backward compatibility issues – reverting the back port.
52. mjsax commented on pull request #7373: KAFKA-7895: Revert suppress changelog bugfix for 2.1 URL: https://github.com/apache/kafka/pull/7373 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to 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