Item 231
git_comments:

1. record two windows worth of sequential values
2. 1MB if values are ms, max is 1000 days
3. ranges is [5000, 15000]
4. 1 MB maximum latency is 10 days; values above that will be pinned
5. e2e latency = 10
6. e2e latency = 20
7. e2e latency = 100
8. e2e latency = 25
9. e2e latency = 15
10. one metric is added automatically in the constructor of Metrics

git_commits:

1. **summary:** KAFKA-9983: KIP-613: add INFO level e2e latency metrics (#8697)
   message: KAFKA-9983: KIP-613: add INFO level e2e latency metrics (#8697) Add e2e latency metrics at the beginning and end of task topologies as INFO-level processor-node-level metrics. Implements: KIP-613 Reviewers: John Roesler <vvcephei@apache.org>, Andrew Choi <a24choi@edu.uwaterloo.ca>, Bruno Cadonna <cadonna@confluent.io>, Matthias J. Sax <matthias@confluent.io>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** KAFKA-9983: KIP-613, add INFO level e2e latency metrics
   body: Moved all metrics to processor-node-level, but the INOF level metrics are recorded only at the source and "sink" nodes (in quotations as this may be a non-sink terminal node)

github_pulls_comments:

1. call for review @guozhangwang @mjsax @vvcephei
2. Retest this please.
3. Alright I fixed the latency measurement to record the right thing @mjsax
4. Actually this needs to be reworked to account for the case there is no sink node. Also got the Percentiles working so I'll add them back to this PR and call for review again when ready
5. **body:** Build failed with checkstyle: ``` [ant:checkstyle] [ERROR] /home/jenkins/jenkins-slave/workspace/kafka-pr-jdk11-scala2.13/clients/src/main/java/org/apache/kafka/common/metrics/stats/Percentiles.java:23:8: Unused import - org.apache.kafka.common.metrics.Measurable. [UnusedImports] ```
   label: code-design
6. Retest this please.
7. Retest this please.
8. spotBugs is currently failing with a mysterious and uninformative exception so there's no use kicking off the tests again until I figure out what's going on. Might be a spotBugs bug ... in local testing I have seen it failing non deterministically... 😒
9. Retest this please.
10. Retest this please.
11. **body:** Test this please
    label: test
12. The only failing tests were unrelated: org.apache.kafka.connect.integration.InternalTopicsIntegrationTest.testCreateInternalTopicsWithFewerReplicasThanBrokers org.apache.kafka.streams.integration.EosBetaUpgradeIntegrationTest.shouldUpgradeFromEosAlphaToEosBeta[false] org.apache.kafka.streams.integration.EosBetaUpgradeIntegrationTest.shouldUpgradeFromEosAlphaToEosBeta[true] Note that the java 8 build is not actually running.
13. Unbelievable. The java 8 build started running right after I mentioned that it's not running.
14. > The java 8 build started running right after I mentioned that it's not running I hope you'll choose to use these new powers for good

github_pulls_reviews:

1. Assuming that a task might have a cache, is this correct, ie, `has been fully processed by the task`)?
2. We we increase this ts to 35? This would allow to test min in the last step better
3. Nice one!
4. Oh right, I put the `record` in the wrong place but this description is correct. It should record at the `RecordCollector` for the task-level metrics

5. **body:** prop (super-nit): Could you call the method `addMinAndMaxToSensor()` since we have already one method that is called `addAvgAndMinAndMaxToSensor()`?
   label: code-design
6. Yes, very nice!
7. req: Please do not use the constant here. The point of this test is also to check the correctness of the description, i.e., the content of that constant.
8. **body:** req: Please define this constant as private as all the others. I left a request in `TaskMetricsTest` which makes this request clearer.
   label: code-design
9. See my comment above.
10. **body:** Q: Is there a specific reason to init the sensor here and not in `SinkNode`? You can init and store it there. That was one motivation to make `*Metrics` classes (e.g. `TaskMetrics`) static, so that you do not need any code in the processor context to get specific sensors. If there is not specific reason, you could get rid of the changes in the `*Context*` classes.
    label: code-design
11. I modified the proposal slightly to make these all processor-node level (will push the changes in a minute) but this question is still relevant, so here's the answer: We can't record the e2e latency in the sink node because not all topologies _have_ a sink node. For that reason we also can't record at the record collector. We need to figure out the terminal nodes when processing the topology, then record this metric after `child.process` in `ProcessorContext#forward`
12. Checkstyle won't allow you to have a letter and number next to each other, but `P90` and `E2E_LATENCY` seem preferable to `P_90` and `E_2_E_LATENCY`
13. **body:** Trying to build confidence in the `Percentiles` implementation and gauge the accuracy with a more complicated test. I found it was accurate to within 5% maybe 2/3 or 3/4 of the time, but it seems reasonable to expect it to be accurate to within 10%
    label: test
14. **body:** Want to call attention to these...do they seem reasonable? The size is the bytes per each percentile sensor, so 2 per source or terminal node. The minimum has to be 0 for the linear bucketing (which I found significantly more accurate than constant bucketing in my tests). On the other hand, the maximum is obviously not representative of the maximum difference between the current and record timestamp. If someone's processing historical data, it can exceed this. But I figure if you're processing historical data than the e2e latency isn't really going to be at all useful anyways, so we may as well set it to something reasonable
    label: code-design
15. Not sure if we might want to make it configurable though? Or just pick a number and see if anyone complains?
16. Let's see...
17. **body:** Not sure about this... Why do we need/want to have a limit? Nit: double space
    label: code-design
18. For this case, should we record "zero" instead?
19. Not sure about this. Why do we need a maximum to begin with? And why pick 10 days? Rather arbitrary?
20. **body:** 10 days was just rounding up from the 7 day default retention limit. The maximum is due to the percentiles calculation which is based on incrementally sized buckets. It's a tradeoff with accuracy For example if I increase it by a factor of 1000, the `StreamTask` percentiles test is off by almost 20% (p99 is 82.9 instead of 99). This test uses values between 0 and 100, which is probably considerably lower than most e2e latencies will be.If you look at the `MetricsTest` percentiles test I added, this uses random values up to the max value and can maintain the 10% accuracy up to a higher max value. Of course we don't know what the distribution will be, but it seems likely to be somewhere in the middle (not in the 100s of ms, not in the 10s or 1000s of days) so for reasonable accuracy we need to pick a reasonable maximum. We can definitely go higher than 10 days, but I reasoned that if you have records earlier than 10 days you're probably processing historical data and in that case the e2e latency isn't that meaningful.
    label: code-design
21. I was debating this...my thinking here was that a negative value probably means you're processing some records with "future" timestamps, for whatever reason, in which case the e2e latency isn't meaningful and they shouldn't affect the statistics. Or, your clocks are out of sync. I suppose we could add a separate metric that counts the number of records with negative e2e latency?
22. Ack (limit explanation on comment below)
23. req: ```suggestion verifySensor(() -> ProcessorNodeMetrics.recordE2ELatencySensor(THREAD_ID, TASK_ID, PROCESSOR_NODE_ID, RecordingLevel.INFO, streamsMetrics)); ```
24. **body:** prop: For the sake of readability, could you extract this check to a method named `isTerminalNode()`? Even better would be to add a method named `isTerminalNode()` to `ProcessorNode` and use it here and in `ProcessorTopology`.
    label: code-design
25. See my comment in `ProcessorContextImpl`.
26. I understand that we need a maximum due to the way the percentiles are approximated. Since the e2e latency depends on user requirements, it would make sense to consider a config for the max latency. I see two reasons for such a config. 1. We always think about near-realtime use cases, but there could also be use cases that are allowed to provide a much higher latency but the latency should still be within a certain limit. For example, one were producers are not always online. Admittedly, 10 days is already quite high. 2. OTOH, decreasing the max latency would also make the metric more accurate, AFAIU. That would also be a reason for a config that users can tweak. For both cases, we could leave it like it is for now and see if there is really the need for such a config. WDYT?
27. @ableegoldman I agree with your thinking here. IMO, we should just log the warning for now. If we see that there is a demand for such a metric, we can add it later on.

28. Q: Wouldn't it be better to count measurements beyond the maximum latency towards the highest bucket as the `Percentiles` metric does? Admittedly, the measured value would be quite wrong in the case of a lot of measurements greater than the maximum latency. However, with the sizes of the buckets that increase linearly, the reported values would be quite wrong anyways due to the increased approximation error. Furthermore, I guess users would put an alert on substantially smaller values. OTOH, not counting measurements beyond the maximum latency would falsify a bit the metric because they would not count towards the remaining 1% or 10% (for p99 and p90, respectively). Additionally, the max metric would also be falsified by not counting those measurements.
29. prop: Take a look into `StreamsTestUtils` and see what you can re-use there to retrieve specific metrics.
30. **body:** req: Please add unit tests in `StreamsMetricsImplTest`.
    label: test
31. Not sure if it really matters, but this is not a uniform distribution (because MAX_VALUE and MIN_VALUE are not integer multiples of 1000 days. If you wanted a uniform distribution, it looks like you can use the bounded `nextInt` and cast to `long`. Also, FYI, `Math.abs(Long.MIN_VALUE) == Long.MIN_VALUE` (which is a negative number), due to overflow.
32. **body:** I'm fine with this as well, although I think it makes more sense either to pin to zero and warn or to just record the negative latency and warn. It feels like we're overthinking it. If the clocks are drifting a little and we report small negative numbers, the e2e latency is still _low_, which is still meaningful information. I really don't see a problem with just naively reporting it and not even bothering with a warning.
    label: code-design
33. Meta-review procedural question: In the future, can we try to avoid making the same comment in multiple places in the PR, since it leads to split discussions like this one?
34. **body:** This necessity makes me think that our Percentiles metric algorithm needs to be improved. Admittedly, I haven't looked at the math, but it seems like it should be possible to be more adaptive. I'm in favor of not adding a config and just leaving it alone for now, so that we can take the option in the future to fix the problem by fixing the algorithm.
    label: code-design
35. However, I *do not* think we should restrict the max value for other metrics than the percentiles one. E.g., there's no reason to restrict the value we record for the max and min metrics. You should be able to update the Percentiles implementation to apply the maximum bound in the metric record method. Otherwise, I'd recommend recording two sensors separately; one for the bounded metrics, and one for the unbounded ones.
36. **body:** If I understand this right, we are recording sink latencies after processing, but source latencies before processing. This nicely avoids the problem with recording non-sink latencies after processing, but is it accurate?
    label: code-design
37. ack
38. I wasn't going for a uniform distribution, just any non-pidgeonholed distribution (see existing `testPercentiles` for comparison). I was just trying to verify the basic validity and get a rough estimate on the accuracy here in case it turned out to be 5000% off. Good point about the overflow though. Pretty annoying that you can't give a bound for `nextLong` like you can with `nextInt` :/
39. We discussed this offline, but in case anyone else was wondering: Yes. We can't record the latency _after_ processing for source nodes due to our recursive DFS approach to processing, as the source node's `#process` actually doesn't complete until the record has been processed by every other node in the subtopology. And anyways, the intent of the source node metric is to gauge the e2e latency when the record arrives at the subtopology, which is what we are recording here.
40. Thanks, @ableegoldman , I think this is a fine tradeoff. Also helping is the fact that we know all "source nodes" are actually instances of SourceNode, which specifically do nothing except forward every record, so whether we measure these nodes before or "after" their processing logic should make no practical difference at all.
41. **body:** Thanks for the details. Avoiding a config for now sounds good to me. This leave the path open to add a config later, or as John suggested to maybe change the algorithm (that might not need a max). I am fine with a hard coded max of 10 days. Also +1 to John's suggestion to split percentiles and min/max to avoid applying the hard coded limed to min/max metric.
    label: code-design
42. I would not record a negative latency. That seems to be kinda weird. I am fine with skipping and warning, too. Just wanted to clarify.
43. > o whether we measure these nodes before or "after" their processing logic should make no practical difference at all. I think it make a big difference, and only recording _before_ processing is what we want (according to what the KIP says). Otherwise, the latency includes the processing time for one or more processors (in the worst case even all processors).
44. **body:** @cadonna @vvcephei @mjsax We have several related discussions going on across this PR so I'm just going to try and summarize here: let me know if I miss anything you still feel is important The plan is to pin large/small values in the percentiles to the min/max for now and just log a warning. Since we're the only users of the `Percentiles` class, we can just modify it directly and avoid restricting the values for the min/max metrics as John mentioned above. If a user is experiencing small negative e2e latencies it's likely due to clock drift, and approximating as 0 seems reasonable. If they're experiencing large negative e2e latencies, there's clearly something odd going on and the e2e latency percentiles aren't meaningful. But it will still show up in the min metric and alert them to this. Presumably users may be interested to know. I'd like to avoid introducing a config in particular because the maximum isn't an inherent mathematical property of percentiles (obviously), it's just an artifact of the existing percentiles algorithm. We can improve this and presumably remove the requirement to set a static max, but I felt the algorithm was "good enough" for now (and didn't want to make large scale changes and/or rewrite it entirely right before the 2.6 deadline). In sum I'd say the guiding principle for this PR and the initial metrics was to be useful without being misleadingly wrong. I think pinning the percentiles to the bounds but reporting the min/max as is achieves this, and allows us flexibility in improving the situation later
    label: code-design

45. **body:** Sorry for my ambiguity. Please let me clarify my terms. Currently if you wait until the end of the "process" method, you wind up including the call to forward, which recursively calls process on all descendents of the source node. This is _not_ what I was talking about. I meant only the time spent _just_ in processing the SourceNode, excluding the time in "forward". What shall we call this? Maybe "actual", or "proper", or "internal" processing time? So, my comment was that, given that we know the implementation of SourceNode, we know that it's "actual", "proper", "internal" processing time is going to be very small, probably far less than a single millisecond. So it doesn't make any practical difference whether we measure before the call for just the special case of source nodes, or magically solve the problem of measuring the e2e latency after internal processing, but not including the calls to "forward". This is why I think it's fine to measure SourceNodes _before_ the call to process, even though the KIP technically specifies that processors' end-to-end latencies should include processing latency. We're making a simplifying assumption that for source nodes specifically, the processing latency would be `<< 1`, so we can ignore it.
    label: code-design

jira_issues:

1. **summary:** Add INFO-level end-to-end latency metrics to Streams
   description: Min and max task-level metrics exposed for all source and terminal nodes

jira_issues_comments: