KIP-963: Additional metrics in Tiered Storage

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: KAFKA-15147 - Getting issue details... STATUS

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

Motivation

Problem

I am an operator of a Tiered Storage Kafka cluster and I would like to know whether interactions with the remote tier, uploads, build auxiliary state, and deletions, are progressing at a steady pace.

For example, if uploads are not progressing at a constant rate I will have data building up in local storage and I might need to take corrective actions (like adding more storage temporarily). Likewise, if deletes are not progressing at a constant rate this might indicate a problem with the retention settings of my topics which I would like to remedy.

Similarly, if there are errors while building the auxiliary state of remote log segments or deleting remote log segments, it could indicate a problem with the Tiered Storage plugins, or the underlying storage.

Solution

To get this observability we would like to expose a remote-upload, remote-delete and other metrics detailed in the table below from the point of view of Kafka. Since upload and deletion are carried out by plugins such metrics emitted by Kafka are going to be **best estimates** on what it believes the state of the world is. We propose the new metrics to be emitted on per topic and per partition granularity.

Other Tiered Storage metrics are emitted only at a topic level, but we would like to also expose these at a partition level because this will allow us to easily track whether a single broker is experiencing a slowdown or the issue is spread across the cluster.

Additionally, we would like to expose remote build auxiliary state error rate and remote delete error rate at a topic level.

We leave more detailed progress metrics to be emitted by the developers of Tiered Storage plugins.

Public Interfaces

Briefly list any new interfaces that will be introduced as part of this proposal or any existing interfaces that will be removed or changed. The purpose of this section is to concisely call out the public contract that will come along with this feature.

MBean	Description
Copy	

kafka.server:type=BrokerTopicMetrics, name=**RemoteCopyLagBytes**, topic=([-.w]+)	The remote copy bytes lag of a topic is defined as the number of bytes in non-active segments eligible for tiering not yet uploaded to the remote storage. In other words, this metric shows the difference between the latest local segment candidate for upload and the latest remote segment in bytes. It shows how the RemoteLogManager is copying the backlog of segments. Normally this lag should be zero, but it will grow when upload is slower than the increase on candidate segments to upload. This metric will be calculated as part of *RemoteLogManager#copyLogSegmentsToRemote*.
kafka.server:type=BrokerTopicMetrics, name=**RemoteCopyLagSegments**, topic=([-.w]+)	The remote copy segments lag of a topic is defined as the number of non-active segments eligible for tiering not yet uploaded to the remote storage. This metric is similar to the above, but it shows the lag in **number of segments** rather than the bytes. This metric will be calculated as part of *RemoteLogManager#copyLogSegmentsToRemote*.
Delete	
kafka.server:type=BrokerTopicMetrics, name=**RemoteDeleteLagBytes**, topic=([-.w]+)	The remote delete bytes lag of a topic is defined as the number of bytes in non-active segments marked for deletion but not yet deleted from remote storage. In other words, this metric shows the difference between latest remote candidate segment to keep based on retention and the oldest remote segment in bytes. If it goes above 0 then it shows that the RemoteLogManager is having a backlog of segments to delete. This metric will be calculated as part of *RemoteLogManager#cleanupExpiredRemoteLogSegments*.
kafka.server:type=BrokerTopicMetrics, name=**RemoteDeleteLagSegments**, topic=([-.w]+)	The remote delete bytes lag of a topic is defined as the number non-active segments marked for deletion but not yet deleted from remote storage. This metric is similar to the above, but it shows the lag in **number of segments** rather than the bytes. This metric will be calculated as part of *RemoteLogManager#cleanupExpiredRemoteLogSegments*.
kafka.server:type=BrokerTopicMetrics, name=**RemoteDeleteRequestsPerSec**, topic=([-.w]+)	The number of delete requests for expired remote segments to remote storage per second. This metric will be calculated as part of *RemoteLogManager#cleanupExpiredRemoteLogSegments*.
kafka.server:type=BrokerTopicMetrics, name=**RemoteDeleteErrorsPerSec**, topic=([-.w]+)	The number of delete requests for expired remote segments to remote storage which resulted in errors per second. This metric will be calculated as part of *RemoteLogManager#cleanupExpiredRemoteLogSegments*.
Fetch	
kafka.server:type=BrokerTopicMetrics, name=**BuildRemoteLogAuxStateRequestsPerSec**, topic=([-.w]+)	The number of requests for rebuilding the auxiliary state for a topic-partition per second. This metric will be incremented whenever there is a successful execution of *ReplicaFetcherTierStateMachine#buildRemoteLogAuxState*.
kafka.server:type=BrokerTopicMetrics, name=**BuildRemoteLogAuxStateErrorsPerSec**, topic=([-.w]+)	The number of requests for rebuilding the auxiliary state for a topic-partition which resulted in errors per second. This metric will be incremented in whenever there is an unsuccessful execution of *ReplicaFetcherTierStateMachine#buildRemoteLogAuxState*.
kafka.server:type=**DelayedRemoteFetchMetrics**, name=**ExpiresPerSec**	The number of expired remote fetches per second. This metric will be incremented in *DelayedRemoteFetch#onExpiration*.
Others	
kafka.server:type=BrokerTopicMetrics, name=**RemoteLogSizeComputationTime**, topic=([-.w]+)	The amount of time needed to compute the size of the remote log. I envision that these three metrics can be calculated as part of handling expiration of log segments.
kafka.server:type=BrokerTopicMetrics, name=**RemoteLogSizeBytes**, topic=([-.w]+)	The total size of a remote log in bytes.
kafka.server:type=BrokerTopicMetrics, name=**RemoteLogMetadataCount**, topic=([-.w]+)	The total number of metadata entries for remote storage.

Proposed Changes

Constraints

The metric calculation should not acquire locks. If it had to acquire locks then it would be contending with the archival/replication/deletion paths of Tiered Storage.

Details

The lag metrics should not only be updated on an archival/deletion cycle. If archiving/deletion is failing to run for whatever reason we should still see the latest state of records being queued up for archival/deletion.

Compatibility, Deprecation, and Migration Plan

These are new metrics and as such shouldn't have compatibility concerns.

Test Plan

Describe in few sentences how the KIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?

Unit and integration tests.

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

1. ~~Emit the lag metrics only on topic level - we rejected this alternative pbecause it wouldn't allow an operator to fairly quickly understand whether any problems are isolated to individual brokers or widespread through the cluster.~~
2. Update the lag metrics on each archival/deletion cycle - we rejected this alternative because if the cycle does not run for whatever reason the old metric would be emitted even if new segments have become eligible for tiering.