KIP-852: Optimize calculation of size for log in remote tier

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *here*

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

Motivation

As per the Tiered Storage feature introduced in KIP-405, users can configure the retention of remote tier based on time, by size, or both. The work of computing the log segments to be deleted based on the retention config is owned by RemoteLogManager (RLM).

To compute remote segments eligible for deletion based on retention by size config, RLM needs to compute the `total_remote_log_size` i.e. the total size of logs available in the remote tier for that topic-partition. RLM uses the `RemoteLogMetadataManager.listRemoteLogSegments()` to fetch metadata for all the remote segments and then aggregate the segment sizes by using `RemoteLogSegmentMetadata.segmentSizeInBytes()`to find the total log size stored in the remote tier.

The above method involves iterating through all metadata of all the segments i.e. O(num_remote_segments) on each execution of RLM thread. Since the main feature of tiered storage is storing a large amount of data, we expect num_remote_segments to be large and a frequent linear scan (i.e. listing all segment metadata) could be expensive/slower because of the underlying storage used by RemoteLogMetadataManager. This slowness could lead to slower rate of uploading to remote tier.

This KIP addresses the problem by proposing a new API in RemoteLogMetadataManager(RLMM) to calculate the total size and delegates the responsibility of calculation to the specific RLMM's implementation. This API removes the requirement to list all segment metadata for calculation of total_size.

(Note: for the case of local storage tier, all log segments are stored in-memory and size is calculated by iterating through the in-memory loaded segments. For remote-tier, we anticipate the number of segments to be significantly larger than local tier segments which might not fit into in-memory cache).

Public Interfaces

New API in RemoteLogMetadataManager (RLMM)

```
/**
* Returns total size of the log for the given leader epoch in remote storage.
*
* @param topicPartition topic partition for which size needs to be calculated.
* @param leaderEpoch Size will only include segments belonging to this epoch.
* @return Total size of the log stored in remote storage in bytes.
*/
long remoteLogSize(TopicPartition topicPartition, int leaderEpoch);
```

New Metric

The following new metrics will be added. RemoteLogSizeBytes will be updated using the values obtained from `remoteLogSize` API call on every attempt to compute remote segments eligible for deletion by the RemoteLogManager.

name	Description
kafka.log.remote:type=BrokerTopicMetrics, name=`RemoteLogSizeBytes`, topic=([-.w]+)	Provides the total size of log in bytes stored on the remote tier.

Proposed Changes

KIP-405 proposes a public interface `RemoteLogMetadataManager` . Users can plugin their own implementation if they intend to use another system to store remote log segment metadata. KIP-405 also provides a default implementation for RLMM called `TopicBasedRemoteLogMetadataManager` which uses topics.

This KIP proposes to delegate the responsibility of calculation of total size of log in remote tier to the specific implementation for RemoteLogMetadataManager To this end, this KIP proposes addition of a new API remoteLogSize to the RLMM interface. RLMM implementations would implement this API and may choose to optimize it based on their internal data structure.

This API would also be useful for other cases such as exposing the amount of data in remote tier for a particular topic partition.

After the implementation of this method, RemoteLogManager would compute the size of log as follows:

```
def calculateRemoteTierSize() {
  // Find the leader epochs from leader epoch cache.
  val validLeaderEpochs = fromLeaderEpochCacheToEpochs(log)
  // For each leader epoch in current lineage, calculate size of log
  val remoteLogSizeBytes = validLeaderEpochs.map(epoch => rlmm.remoteLogSize(tp, epoch)).sum
  remoteLogSizeBytes
}// the new  API would be used for size based retention as:

val totalLogSize = remoteLogSizeBytes + log.localOnlyLogSegmentsSize

var remainingSize = if (shouldDeleteBySize) totalLogSize - retentionSize else 0

val segmentsIterator = remoteLogMetadataManager.listRemoteLogSegment

while (remainingSize > 0 && segmentsIterator.hasNext) { // delete segments }
```

Code changes

1. Add the new API to `RemoteLogMetadataManager`
2. Implement the new API at `TopicBasedRemoteLogMetadataManager` (with unit tests)
3. Add the new metric when code for RemoteLogManager has been merged.

Implementation at TopicBasedRemoteLogMetadataManager

TopicBasedRemoteLogMetadataManager keeps the metadata stored in a cache `RemoteLogMetadataCache`, hence, iterating through the list of all segments to compute total size would not be computationally expensive. We would keep the existing logic of computation for total log size which is based on `RemoteLogMetadataManager.listRemoteLogSegments` but move it inside `TopicBasedRemoteLogMetadataManager.`

Metrics

This KIP proposes to add a new metric `RemoteLogSizeBytes` which tracks the size of data stored in remote tier for a topic partition.
This metric will be useful both for the admin and the user to monitor in real time the volume of the more tiered data. It would be used in future to add the size of remote tier in response to `DescribeLogDirs` API call. RemoteLogSizeBytes will be updated using the values obtained from `remoteLogSize` API call each time we run the log retention check (that is, log.retention.check.interval.ms) and when user explicitly call remoteLogSize().

Compatibility, Deprecation, and Migration Plan

Introduction of RemoteLogManager is under review at https://github.com/apache/kafka/pull/11390 and the capability to delete has not been implemented yet. Hence, addition of this new API in RemoteLogMetadataManager(RLMM) will not impact any existing code since the RLMM is not being used anywhere in the existing code.
A default implementation of RLMM is available as `TopicBasedRemoteLogMetadataManager`. As part of this KIP, a concrete implementation of the API will be added to TopicBasedRemoteLogMetadataManager. It will not have any compatibility impact in existing code since TopicBasedRemoteLogMetadataManager is not being used anywhere.

Test Plan

Tests will be added to RemoteLogMetadataManagerTest and TopicBasedRemoteLogMetadataManagerTest relevant to the code changes mentioned above.

Rejected Alternatives

Alternative 1: Delegate the responsibility to find retention breaching segments to RLMM

Pros: Allows for deeper optimisations by the RLMM implementation.
Cons:

1. Less generally useful, a potentially trivial optimisation.
2. This solution adds additional responsibility to the plugin, hence adding more complexity & effort towards plugin development.

Alternative 2: Modify RemoteLogMetadataManager.`listRemoteLogSegments` to provide iteration from tail of the log

This approach advocates for removing the need for calculating total size of the log. The deletion process would be changed to start iterating from the tail of the log. The iteration will continue until the configured retention size and segments beyond that point will be eligible for deletion.

Pros: Simplifies the deletion logic without having to calculate the total size of the log.
Cons:

1. RLMM implementation has an additional responsibility to list metadata in decreasing order of offsets. This adds an additional requirement for the underlying implementation of RLMM to perform this sort which might not be optimised when dealing with a large number of segments.
2. Metric to track the total size of remote tier will still need an implementation of the new API `` `remoteLogSize()` ``
3. We would need to iterate through the list of segments which are not eligible to be deleted. This could be an expensive operation if we do it on every deletion.

Alternative 3: Store the cumulative size of remote tier log in-memory at RemoteLogManager

This approach advocates for maintaining the size of log in remote tier in-memory and updating it every time there is a copySegmentToRemote or a deleteSegment event. The in-memory value needs to be initialised once by performing a full scan of all log segments, typically at broker startup.

Pros: Constant time calculation of size since it is stored in-memory.
Cons: Every time a broker starts-up, it will scan through all the segments in the remote tier to initialise the in-memory value. This would increase the bootstrap time for the remote storage thread pool before the first eligible segment is archived.

Alternative 4: Store the cumulative size of remote tier log at RemoteLogManager

This approach improves on the disadvantages of Alternative 3 by storing the size in a persistent storage. For the choice of persistent storage, there is no prior example in Kafka on storing metrics such as this.
We could choose to:

1. persist this metric in control plane metadata log or
2. introduce a new reserved log compacted topic or
3. persist the cumulative total size of each topic-partition in a file or
4. use in-memory b-tree/lsm tree backed my persistent storage

In 1, we risk coupling the remote tier feature with other Kafka logic which KIP-405 explicitly tries to avoid.
In 2, adding another reserved topic for metric may not be viable given the overhead of having a reserved topic.
In 3, for efficient look up, we would need to introduce an index for the file which stores the offsets for each topic partition and leader epoch. Additionally, we would incur IO latency cost since the update would require a look-up from the file and writing to the file on every copyToRemote or deleteRemoteSegment operation.
In 4, overhead of maintaining a b-tree might be an overkill for this feature.