KIP-1002: Fetch remote segment indexes at once

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: "In Discussion"

Discussion thread: https://lists.apache.org/thread/jfj4fn09v5d1n9s7vpt1twht9jodfvlk

JIRA: Unable to render Jira issues macro, execution error.

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

Motivation

In Tiered Storage, indexes are uploaded together with the segment log (on the same operation `RemoteStorageManager#copyLogSegment`) when segments become available for uploading to the remote tier.

However, when consumer fetching hits remote segments then each segment index is fetched individually and cached in memory at the broker. Even though fetching segment indexes is an action that happens at least once per segment read, the fact that each segment index is fetched individually can increase latency as fetching cross segments. This can become a more pressing issue when the segment sizes are smaller than the default (e.g. 10s of MBs).

Remote Storage Manager implementations (referred to as Plugins) can optimize how segment indexes are uploaded given that all indexes are uploaded on the same operation, and are immutable. For instance, plugins can choose to upload the segment indexes as part of the same object and fetch them by range.

Nevertheless, fetching segment indexes individually is more expensive given that remote fetch round-trip is comparatively the most expensive operation in terms of latency and carries a (usually small, but additional) cost.

To reduce the impact on latency and the number of calls to remote objects, this KIP proposes to include new operations to the `RemoteStorageManager` API to fetch indexes at once allowing plugin implementations to optimize the number of objects, number of requests and latency.

Public Interfaces

Add the following method to `RemoteStorageManager` API:

```
    /**
     * Returns a set of indexes available for a log segment.
     * If the index type is not available (e.g. transaction index), the resulting map will not include it.
     */
    default Map<IndexType, InputStream> fetchIndexes(RemoteLogSegmentMetadata remoteLogSegmentMetadata,
Set<IndexType> indexTypes) throws RemoteStorageException {
        Map<IndexType, InputStream> indexes = new HashMap<>(indexTypes.size());
        for (IndexType indexType: indexTypes) {
            try {
                indexes.put(indexType, fetchIndex(remoteLogSegmentMetadata, indexType));
            } catch (RemoteResourceNotFoundException e) {
                // log
            }
        }
        return indexes;
    }


    Set<IndexType> INDEX_TYPES = Arrays.stream(IndexType.values()).collect(Collectors.toSet());

    /**
     * Returns all indexes available for a log segment
     * If the index type is not available (e.g. transaction index), the resulting map will not include it.
     */
    Map<IndexType, InputStream> fetchAllIndexes(RemoteLogSegmentMetadata remoteLogSegmentMetadata) throws
RemoteStorageException {
        return fetchIndexes(remoteLogSegmentMetadata, INDEX_TYPES);
    }
```

Both APIs are proposed for current and future tooling and versions of the Tiered Storage framework optimize how indexes are fetched.

Proposed Changes

Apart from including the new APIs, this proposal includes refactoring `RemoteIndexCache` to work with the new API:

The remote index cache has an in-memory map between Segment IDs and a set of indexes (i.e. Entry) containing Offset, Timestamp, and Transaction Index if exists. The cache is backed by files used to maintain the cache across restarts.

Instead of fetching each individual index whenever a file is not found locally, if the entry is invalid (e.g. no files found, not all files found, corrupted files) then the proposed operation to fetch a set of index types is used to get all indexes at once, persist them, and load the cache.

Compatibility, Deprecation, and Migration Plan

To be backward compatible, a default implementation mimicking the current behavior is provided.

Test Plan

The new methods will have to be implemented and tested by the RSM implementations. As `RemoteIndexCache` implementation will be adapted to the new APIs, the current set of tests should serve as a validation that functionality isn't broken.

Tiered Storage integration tests will be extended to cover the new APIs and test it similarly to the existing ones.

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.

Cache Indexes at the RSM Plugin level

RSM implementations could cache indexes at the plugin level, so if the next request is for another index of the same segment, it can be served from the cache.

Although this solves the issue with multiple requests to the same object, it duplicates the effort of keeping 2 caches for indexes, one at the broker level, and one at the plugin level.

As currently this is managed by the broker, and unless this responsibility is delegated (if needed) to the plugin, then the proposed approach stands.