KIP-263: Allow broker to skip sanity check of inactive segments on broker startup

Status

Current state: *Discarded. Withdrawn because the solution we agreed on does not require interface change.*

Discussion thread:

JIRA:

Motivation

Currently broker will sanity check index files of all log segments in the log directory after the broker is started and before it becomes leader or follower for any partition. Sanity check a index file can be slow because broker needs to open index file and read data into page cache. Even if we apply time-based retention policy, the number of inactive log segments will still increase as the usage (e.g. number of partition and total-bytes-in-rate) of the Kafka cluster increases. As a result, the time to restart a broker will increase proportional to the number of segments in the log directory.

The problem with prolonged broker start time is that it will take a long time to rolling bounce a large cluster in order to deploy a new config or hotfix a config or bug. In one of our cluster, we observed that each broker spends around 8 minutes on LogManger.loadLogs(). And then the next broker needs to wait more than 8 minutes for URP to drop to 0 before it can shutdown itself. For a large cluster with 30 brokers, the total time to rolling bounce the cluster will be (8+8)*30 = 360 minutes, or 8 hours.

Ideally we would like broker startup time not to be proportional to the number of inactive segments. Most inactive segments will not be consumed, and even if it is consumed, its index files most likely will not require recovery after clean broker shutdown. This KIP proposes a way to lazy load log segments in order to reduce the cluster rolling bounce time.

Public Interfaces

Add broker config sanity.check.all.logs.enabled. This config value will be true by default.

If it is set to true, sanity check all log segments on broker startup. Otherwise, sanity check only the active log segments on broker startup. Inactive segments will be sanity checked after broker becomes leader for those partitions, either by the background threads, or by the request handler thread if the log segment has not been sanity checked when the request handler thread tries to access the log segment.

Proposed Change

This KIP proposes the following changes:

1) When we instantiate LogSegments and AbstractIndex instances, we skip the instantiation of variables that require disk access, such as MappedByteBuffer, length etc. We will instantiate these variables when we actually need their values. Thus the instantiation of these objects will be much faster. This change applied regardless of the broker configuration.

2) If sanity.check.all.logs.enabled is set to false, then broker does the following on startup:

 - During LogManager.loadLogs(), if there is cleanshutdown file, then the broker only sanity check the active segments. It does not sanity check inactive segments and thus the broker can finish log loading quickly.

 - After broker receives LeaderAndIsrRequest, broker schedules background threads to sanity check the inactive log segments. This allows broker to catch problem soon after it becomes leader for partitions. Note that this may reduce the throughput when producer starts to produce to these partitions, because these background threads will incur a lot of IO access and also grab the lock for each Log during sanity check. Because the byte-in-rate is smaller, URP should drop to zero faster, and the next broker should be able to start shutdown sooner, which also help reduces the rolling bounce time.

- If there is bootsrap consumer to consume from inactive segments before the background thread finishes sanity check the inactive segment, the request handler thread will sanity check the inactive segment.

- If there is log corruption in any segment, all segments prior to this segment will be sanity checked as well before any other thread can access log segments or index files of this partition.

Here are the caveats in delaying the sanity check which we should be aware of:

1) After the active segment passes sanity check, we allow clients to produce/consume from the active segment. But if there is corruption in the inactive segments of this partition, the response to client may be wrong when broker performs transaction-related operation.

Solution: This is not currently an issue if user does not use transaction. Maybe we should also sanity check all log segments of a partition if there exists transaction-related data for this partition.

2) We only sanity check the inactive segments of a partition asynchronously on becoming leader for those partitions. During this period, if a fetch request arrives on a portion of the log that is yet to be sanity checked, then the request handler thread will sanity check that segment in the request handler, which can take some time if there is data corruption. More request handler thread will be blocked on sanity check if they try to access the log segments of these corrupted partition.

Solution: At LinkedIn we currently require client to tolerate at least 120 seconds of unavailability (with 20 retries and 10 seconds retry backoff) which will happen during leadership transfer. This should be sufficient for sanity check if there is no log corruption. Log corruption after clean broker shutdown is very rare. If there is log corruption for many log segments after clean shutdown, most likely there is hardware issue and it will likely affect the active segment as well. If there is log corruption in the active segment, we will sanity check all segments of this partition and therefore broker degrates to the existing behavior, which should avoid the concern that otherwise can happen if we only sanity check after broker becomes leader. So the probability of this becoming an issue should be very small.

Evaluation

We run test in an experiement cluster of 15 brokers on 15 machines. There 31.8k partitions with RF=3 which are evenly distributed across brokers. Each broker has roughly 3000 log segments. Total bytes-in-rate is around 400 MBps.

Here is the evaluation result:

- On a given broker in the test cluster, LogManger startup time reduced from 311 sec to 15 sec.
- When doing rolling bounce in the test cluster, rolling bounce time reduces from 135 minutes to 55 minutes.
- When there is no log corruption, the maximum time to sanity check a partition across all partitions in the test cluster is 59 seconds. If all index and timeindex files of this partition are deleted, the time to recover this partition is 265 seconds.

Compatibility, Deprecation, and Migration Plan

This KIP has no compatibility concern.

Rejected Alternatives

Future work