KIP-16 - Automated Replica Lag Tuning

- Status
- Motivation
- Public Interfaces
- Proposed Changes
    - Expected Scenarios
- Rejected Alternatives

Status

Current state: *Accepted*

Discussion thread: here

JIRA: KAFKA-1546

Released: 0.8.3

Motivation

Currently, replica lag configuration cannot be tuned automatically for high and low volume topics on the same cluster since the lag is computed based on the difference in log end offset between the leader and replicas i.e. number of messages. The default is 4000 messages. For high volume topics, producing even a single large batch can cause replicas to fall out of ISR and in the case of low volume topics detecting a lagging replica takes a very long time. We need a consistent way to measure replica lag in terms of time.

Public Interfaces

This proposal removes 1 config and changes the meaning of another config.

replica.lag.max.messages - This config is deleted since this proposal no longer measures replica lag in terms of number of messages

replica.lag.time.max.ms - The definition of this config now changes. If a follower hasn't sent any fetch requests for this window of time, the leader will remove the follower from ISR (in-sync replicas) and treat it as dead. In addition, if a replica is has not read from the log end offset for this time, it is deemed to not be in ISR because it is not caught up.

Proposed Changes

The proposal is to calculate replica lag as the amount of time not caught up to the leader. A replica is deemed to be "*caught up"* if it's last fetch request read up to the log end offset of the broker at that time and if it made a fetch request within the last n units of time. A replica is only in ISR if it is caught up.

On each replica fetch request, the broker will calculate if the request read from the log end offset i.e. the most recent data that the broker has. Each Replica object maintains a lagBegin metric which is a timestamp corresponding to when the replica first started "lagging". Here's how the code is structured:

```
// Replica.scala
val readToEndOfLog = logReadResult.initialLogEndOffset.messageOffset - logReadResult.info.fetchOffset.
messageOffset <= 0
  if(! readToEndOfLog) {
    lagBeginValue.compareAndSet(-1, time.milliseconds)
  } else {
    lagBeginValue.set(-1)
  }
}

def lagBeginTimeMs = lagBeginValue.get()

// Partition.scala - This is how we calculate slow replicas
val slowReplicas = candidateReplicas.filter(r => (r.lagBeginTimeMs > 0 && (time.milliseconds - r.
lagBeginTimeMs) > keepInSyncTimeMs))
```

lagBeginValue < 0 means that the replica is not lagging. There is a subtle case where data is read from the log end offset and an append occurs immediately afterwards but before the lag calculation is complete. To make this more robust, the LogReadResult should return the LogOffsetMetadata corresponding to the LEO just before the read started.

Expected Scenarios

1. Lagging follower - Follower for any high volume topic does not stay in ISR only if it cannot read from the log end offset for replica.lag.max.ms time. Previously it would fall out of ISR even if a single large batch was produced

2. Stuck Follower - As before, if a follower does not make a fetch request for a period of time, it shall be removed from ISR

3. Bootstrapping follower - If a follower is down for an extended period of time, it will stay outside of the ISR until it is caught up.

I plan to test all these scenarios on a cluster.

Compatibility, Deprecation, and Migration Plan

This change is fully backward compatible. The only difference is that customers will no longer have to set the replica.lag.max.messages config.

Rejected Alternatives

Time based Replica Lag detection - In this approach, we can calculate the replica lag as the estimated amount of time the replica will need to catch up to the leader. However this requires us to associate a commit timestamp with each message on the master. Such metadata does not exist and is out of scope of this proposal. We can also attempt to calculate the same using the current message throughput per-partition. This approach is also flawed because a low-volume topic can be thrown out of ISR by a burst of traffic even if the replica is not falling behind (read JIRA for more details)