KIP-501 Avoid out-of-sync or offline partitions when follower fetch requests are not processed in time

Status

Current State: "*Discussion"*

Discussion Thread: here, here

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

Motivation

Follower replicas fetch data from the leader replica continuously to catch up with the leader. Follower replica is considered to be out of sync if it lags behind the leader replica for more than *replica.lag.time.max.ms*

- If a follower has not sent any fetch requests within the configured value of *replica.lag.time.max.ms.* This can happen when the follower is down or it is stuck due to GC pauses or it is not able to communicate to the leader replica within that duration.
- If a follower has not consumed up to the leader's log-end-offset at the earlier fetch request within *replica.lag.time.max.ms.* This can happen with new replicas that start fetching messages from leader or replicas which are kind of slow in processing the fetched messages.

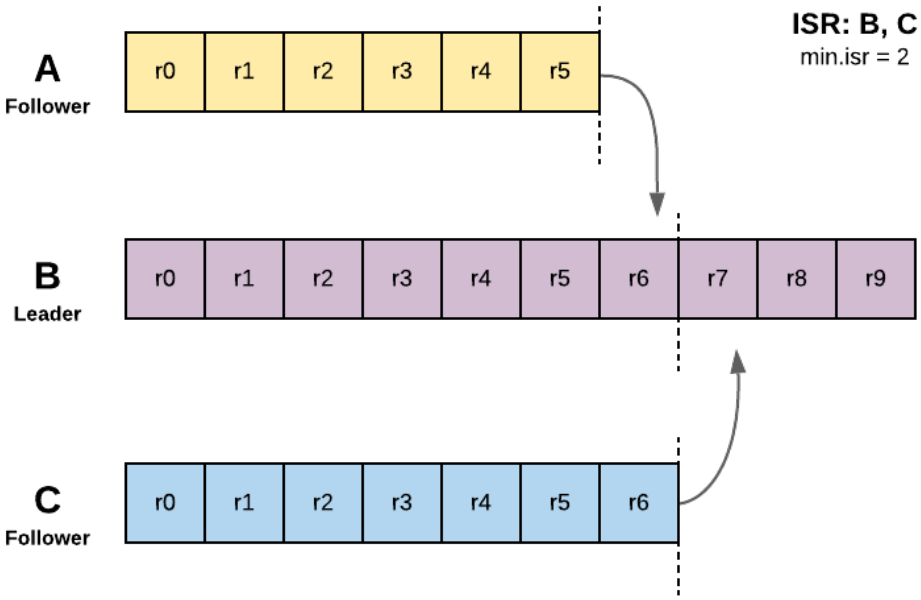
But we found an issue of partitions going offline even though follower replicas try their best to fetch from the leader replica. Sometimes, the leader replica may take time to process the fetch requests. This can be due to having issues in memory or disk while reading the respective segment data or processing fetch requests because of intermittent CPU issues. it makes follower replicas out of sync even though followers send fetch requests within *replica.lag.time.max.ms* duration. We observed this behavior multiple times in our clusters.

It leads to one of the below two scenarios

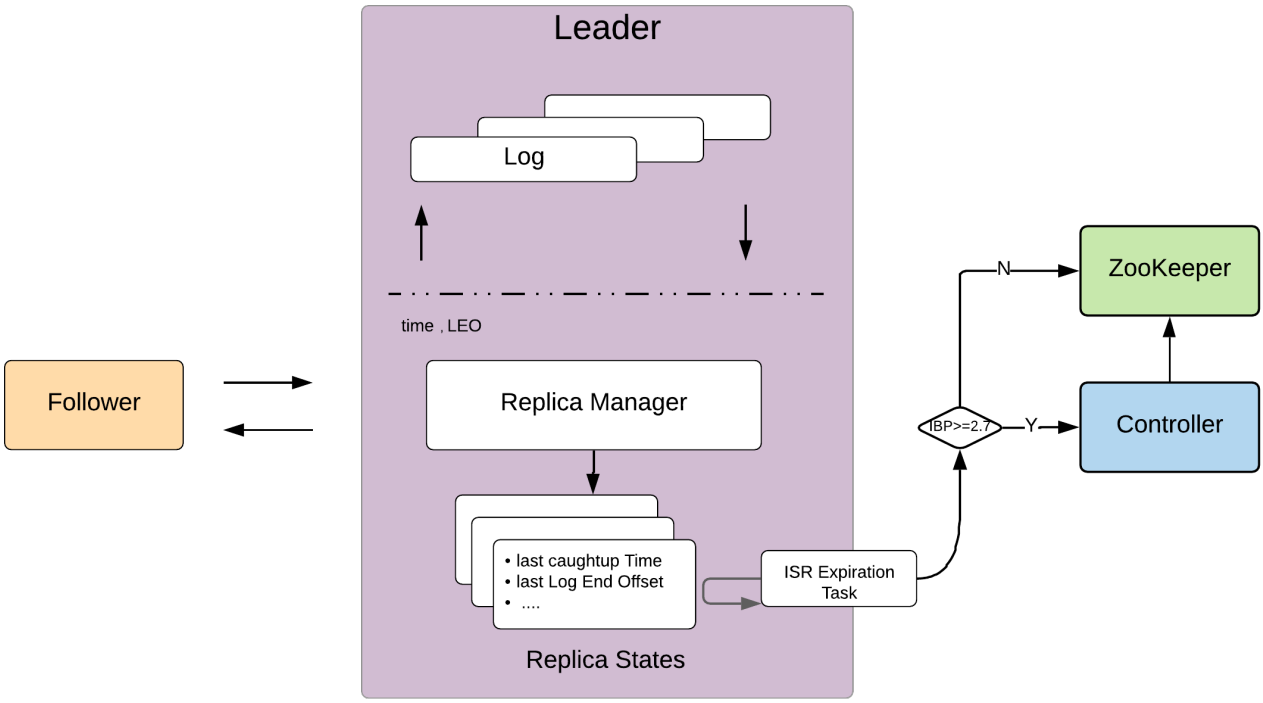
- When in sync replicas go below *min.insync.replicas* and leader can not serve produce requests successfully with acks=all as it required replicas are out of sync.
- Causing offline partitions when in-sync replicas go below *min.insync.replicas* count and the current leader goes down. In this case, other followers can not be leaders.

Proposed Changes

Follower replicas send fetch requests to leader replica to replicate the messages for topic partitions. They send fetch requests continuously and always try to catchup leader's log-end-offset.



replica.fetch.wait.max.ms property represents wait time for each fetch request to be processed by the leader. This property value should be less than the *replica.lag.time.max.ms* so that a follower can send the next fetch request before a follower replica is considered out of sync. This will avoid frequent shrinking of ISRs. This allows a follower replica always tries its best to avoid going out of sync by sending frequent fetch requests within *replica.lag.time.max.ms.*



Each broker runs a ISR update task("isr-expiration") at regular intervals to update the metadata in ZK(directly or via controller if IBP version >= 2.7) about insync replicas of all the leader partitions of this broker. Its interval is *0.5 * replica.lag.time.max.ms*. This allows an insync follower replica can lag behind the leader replica upto *1.5 * replica.lag.time.max.ms*.

Leader replica maintains state of each follower replica that includes

- Followers log start offset
- Followers log end offset
- Leader log end offset at the last fetch request

- Last fetch time
    - Fetch request time of the earlier fetch request. This is used in computing last caught up time.
- Last caught up time

LastCaughtupTime represents the time at which a replica is considered caught up with the leader. This is set as the fetch request time in the leader partition before the request is served successfully.

LastCaughtUptime is updated when the follower's fetch offset

- >= current leader's log-end-offset or
- >= leader's log-end-offset when the earlier fetch request was received.

When a follower replica has the same log-end-offset as leader, then it is considered as insync replica irrespective of whether there are follower fetch requests within the replica lag max time or not.

Replica is considered out of sync only if *(Now - lastCaughtUpTime) > replicaLagTimeMax* and its log-end-offset is not equal to leader's log-end-offset.

One way to address this issue is to have the below enhancement while considering a replica insync

- Fetch requests are added as pending requests while processing the fetch request in `ReplicaManager` if the fetch request's message-offset >= leader's LEO when the earlier fetch request is served. This replica is always considered to be insync as long as there are pending requests. This constraint will guard the follower replica not to go out of sync when a leader has some intermittent issues in processing fetch requests(especially read logs). This will make the partition online even if the current leader is stopped as one of the other insync replicas can be chosen as a leader. Pending fetch request will be removed once the respective fetch request is completed.
- Replica's `*lastCaughtUptime*` is updated with the current time instead of the fetch time in `*Replica#updateFetchState*` based on the fetch offset.

This feature can be enabled by setting the property *"follower.fetch.pending.reads.insync.enable"* to true. The default value will be set as false to give backward compatibility.

This approach will reduce the offline partition occurrence. But the main disadvantage is it can still happen when there are requests queued up and the existing fetch requests io threads are taking longer. The subsequent requests may get stuck in the queue and they may not be able to get served before the ISR Expiration Task considers them out of sync and eventually causes offline partitions.

Solution 2

This is an extension of Solution 1 with the leader relinquishing the leadership if a fetch request takes longer than expected. It will also move its broker id to last in the sequence of ISRs while sending AlterISRRequest to Zookeeper or Controller. That will avoid choosing this broker as the leader immediately by the controller. This approach will mitigate the case of requests getting piled up in the requests queue as mentioned earlier. We can introduce the respective config for the timeout with a default value.

The main disadvantage with this approach is ISR thrashing may occur when there are intermitten issues across brokers.

Example :

<Topic, partition>: <events, 0>

replica.lag.max.time.ms = 10_000

follower.fetch.process.time.max.ms = 500

Assigned replicas: 1001, 1002, 1003, isr: 1001,1002,1003, leader: 1001

Follower fetch request from broker:1002 to broker:1001

At time t = tx , log-end-offset = 1002, fetch-offset = 950

At time t = ty , log-end-offset = 1100, fetch-offset = 1002, time taken to process the request is: 25 secs

Any in-sync replica check after tx+10s for events-0 would result in false that makes the replica on 1002 as out of sync. Replica on 1003 may also get out of sync in a similar way.

With this feature enabled:

While the fetch request is being processed, insync replica checks on the leader replica return true as there are pending fetch requests with fetch offset >= LEO at earlier fetch request.

Once the fetch request is served, lastCaughtupTime is set as ty+25. Subsequent insync checks would result in true as long as the follower replica sends fetch requests at regular intervals according to replica.lag.time.max.ms and replica.fetch.wait.max.ms. Leader will also lose the leadership and one of the insync replicas becomes a leader.

Public Interfaces

Below property is added to the broker configuration.

Configuration Name	Description	Valid Values	Default Value
follower.fetch.pending.reads.insync.enable	This property allows a replica to be insync if the follower has pending fetch requests which are >= log-end-offset of the earlier fetch request.	true or false	false
follower.fetch.process.time.max.ms	This property represents the maximum time in milliseconds that a leader can process the follower fetch requests. If it takes more than this configured time then it relinquishes the leadership. This property is applicable only if `*follower.fetch.pending.reads.insync.enable*` is set as true.	> 0	500

Compatibility, Deprecation, and Migration Plan

This change is backward compatible with previous versions.