Request Purgatory (0.8)

Request Purgatory (0.8)

Without going into too much details (the code holds the truth), this page describes the work done by the request purgatory and its associated components.

What is purgatory?

The request purgatory is a holding pen for requests waiting to be satisfied (*Delayed*). Of all Kafka request types, it is used only for Produce and Fetch requests. The main reasons for keeping requests in purgatory while waiting for them to be satisfied are:

- support for long polling fetch requests; e.g. it keeps clients from issuing repetitive requests to fetch messages when no data is readily available
- avoid blocking the server network threads and filling the request queue while waiting until conditions to send producer and fetch responses are met

Both produce and fetch request types have different conditions to be added to and removed from their respective purgatory. In fact, there is a requests purgatory implementation for Producer (ProducerRequestPurgatory) and Fetch (FetchRequestPurgatory). Both extend the RequestPurgatory abstract class to add request type-specific implementations for expiration and satisfaction conditions checks.

Also, it should be noted that messages are not dropped by the purgatory but simply removed from its watch when they are satisfied. A client will always get a response from the broker under normal conditions, no matter if everything went smoothly or if there was an error.

Flow

When a request comes in and falls under the conditions to be put in purgatory (delayed), the following happens:

1. request counter incremented
    - this counts the number of requests added to the purgatory since the last purge
    - it is the counter used with the **.purgatory.purge.interval.requests* configuration
2. request added to watchers pool with key based on (topic, partition) tuple.
    - the watcher serves to check if the work mandated by the request has been completed.
    - every request coming in for the same tuple will trigger a check of all requests (this piece of information is important to explain the potential OOM problem and the purge interval configuration).
3. enqueued (DelayQueue) for the request reaper thread (detailed below)

So, there are 2 references of the delayed requests, one with the purgatory's watcher and another one for the request reaper. Once the request is satisfied, it will eventually be removed (purged) from both components.

Request Reaper Thread

The requests reaper thread purpose is to expire requests that have been delayed past their deadline.

It polls for expired requests and, when it finds one, runs the purgatory's expire method to handle the delayed request expiration. In both produce and fetch cases, it sends a response to the client. An expired request will be a satisfied request so it is eventually purged from both components.

The next step of the thread's loop is when it checks for the configuration parameters (**.purgatory.purge.interval.requests*). When the number of delayed requests given to watch by the purgatory reaches this value, it goes through all previously queued requests and removes those which are marked as satisfied. Because of that, it is really an interval more than it is a threshold since it doesn't really care about the amount of satisfied requests or the size of the queue.

Purge interval configuration

The purge interval configuration (**.purgatory.purge.interval.requests*) is mostly an "internal" configs that generally don't need to be modified by users. The reasons why it was added are as follow:

- We found that for low-volume topics, replica fetch requests were getting expired but sitting around in purgatory
- This was because we were expiring them from the delay queue (used to track when requests should expire), but they were still sitting in the watcherFor map - i.e., they would get purged when the next producer request to that topic/partition arrived, but for low volume topics this could be a long time (or never in the worst case) and we would eventually run into an OOME.
- So we needed to periodically go through the entire watcherFor mapand explicitly remove those requests that had expired.

> More details on this are in KAFKA-664

Producer requests handling

When is it added to purgatory (delayed)?

- When it uses ack=-1 (actually, anything but 0 or 1),
- Partitions have more than one replica (in this case, ack=-1 isn't different to ack=1 and it doesn't make much sense to use a delayed request),
- Not all partitions are in error

Producer config: *request.required.acks*

When does it expire?

When it reaches the timeout defined in the produce request (ackTimeoutMs). Translates from producer config *request.timeout.ms*.

What happens (on the broker) when it expires?

Sends a response to the client. Response content depends on the request of course.

When is it satisfied?

In a few words, it will be satisfied when enough followers have reached the offset required, based on ack value (-1/all or number of replicas in ISR)

Fetch requests handling

When is it added to purgatory (delayed)?

A fetch request will be added to purgatory depending on the following requests parameters:

- MaxWaitTime is 0 (or less)
- MinBytes is greater than the number of bytes readily available to fullfil the request

Corresponding consumer configurations:

- MaxWaitTime: *fetch.wait.max.ms*
- MinBytes: *fetch.min.bytes*

When does it expire?

After the amount of time the consumer is willing to wait for data (MaxWaitTime).

Consumer configuration: *fetch.wait.max.ms*

When is it satisfied?

A fetch request is satisfied when it has accumulated enough data to meet the MinBytes field

Consumer configuration: *fetch.min.bytes*

Metrics

The following metrics exists for fetch and produce purgatories:

- PurgatorySize (Gauge): used to keep a tab on the number of requests sitting in purgatory (including both watchers map and expiration queue), mostly used as a rough gauge of memory usage
- NumDelayedRequests (Gauge): count of number of requests queued by the request reaper thread (under watch for expiration). It should present a most accurate view of the number of request in purgatory as one point.

To be verified
The same metric name is used for both Fetch and Producer, I should probably open a Jira for this but it gives trouble to the csv reporter since it tries to create a second file with the same name (for whichever metric comes first)

References

- Reason behind clean up interval, discussion in Kafka server threads die due to OOME during long running test
- Original user mailing list email that triggered the creation of this page