KIP-580: Exponential Backoff for Kafka Clients

Status

Current state: *"Accepted"*

Discussion thread: part 1: *here*, part 2*: here*

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

One of the most common problems that we have in Kafka is with respect to metadata fetches. For example, if there is a broker failure, all clients start to fetch metadata at the same time and it often takes a while for the metadata to converge. In a high load cluster, there are also issues where the volume of metadata has compounded this problem and made convergence of metadata even slower. The reason for this is because we have a static retry backoff mechanism. However, with a small backoff (currently the default is 100 ms), we could send tens of requests per second to the broker, and if the connection issue is prolonged, this would lead to the problem described previously. To reduce this pressure, it would be useful to support an exponentially increasing backoff policy for all Kafka clients, similar to the configuration introduced in KIP-144 for exponentially increasing backoffs for broker reconnect attempts.

Public Interfaces

We can introduce a new common client configuration for clients called `retry.backoff.max.ms` that is defined as:

> *The maximum amount of time in milliseconds to wait when retrying a request to the broker that has repeatedly failed. If provided, the backoff per client will increase exponentially for each failed request, up to this maximum. To prevent all clients from being synchronized upon retry, a randomization factor of 0.2 will be applied to the backoff, resulting in a random range between 20% below and 20% above the computed value. If retry.backoff.ms is set to be higher than retry.backoff.max.ms, then retry.backoff.max.ms will be used as a constant backoff from the beginning without any exponential increase.*

`retry.backoff.max.ms` would default to 1000 ms, and the starting backoff will be derived from `retry.backoff.ms` (which defaults to 100 ms). If `retry.backoff.ms` is set to be greater than `retry.backoff.max.ms`, then `retry.backoff.max.ms` will be used as the backoff. Also, if `retry.backoff.ms` is set to be greater than `retry.backoff.max.ms`, then we will log a warning. The formula to calculate the backoff is as follows:

```
MIN(retry.backoff.max.ms, (retry.backoff.ms * 2**(failures - 1)) * random(0.8, 1.2))
```

"random" in this case is the random function that will randomly factor in a "jitter" that is 20% higher or lower to the computed value. This will keep increasing until it hits the `retry.backoff.max.ms` value.

Ignoring the randomized jitter for the sake of the example below, we can see the following example to illustrate the benefits of exponential backoff in comparison to static backoff. Currently, with the default of 100 ms per retry backoff, in 1 second we would have 10 retries. In the case of using an exponential backoff, we would have a total of 4 retries in 1 second. Thus we have less than half of the amount of retries in the same timeframe and can lessen broker pressure. This calculation is done as following:

Try 1 at time 0 ms, failures = 0, next retry in 100 ms (default retry ms is initially 100 ms)
Try 2 at time 100 ms, failures = 1, next retry in 200 ms
Try 3 at time 300 ms, failures = 2, next retry in 400 ms
Try 4 at time 700 ms, failures = 3, next retry in 800 ms
Try 5 at time 1500 ms, failures = 4, next retry in 1000 ms (default max retry ms is 1000 ms)

Proposed Changes

Since there are different components that make use of the `retry.backoff.ms` configuration, each module that uses this will have to be changed. That being said, across all components, there will be a similar logic for dynamically updating the retry backoff, just making it fit to the component it's in.

Admin Client

In `KafkaAdminClient`, we will have to modify the way the retry backoff is calculated for the calls that have failed and need to be retried. From the current static retry backoff, we have to introduce a mechanism for all calls that upon failure, the next retry time is dynamically calculated.

Consumer

In `KafkaConsumer`, we have to change the retry backoff logic in the `Fetcher`, `ConsumerNetworkClient`, and `Metadata`. Since `ConsumerNetworkClient` and `Metadata` are also used by other clients, they would have to house their own retry backoff logic. For the `Fetcher` however, it could query a dynamically updating `retryBackOffMs` from `KafkaConsumer`.

Producer

For the `KafkaProducer`, we have to change the retry backoff logic in `ConsoleProducer, RecordAccumulator, Sender, TransactionManager,` and `Metadata`. As mentioned above, `Metadata` is used by other clients, so it would have its own retry backoff logic. For the rest of the classes, as described in the "Consumer" section above, it can query a dynamically updating `retryBackOffMs` from `KafkaProducer`.

Broker API Versions Command

Changes made for `AdminClient` and `ConsumerNetworkClient` would apply here. The main changes that would have to be made are for `BrokerApiVersionsCommand` to set the appropriate arguments for `AdminClient` and `ConsumerNetworkClient` after changes for those are made.

Compatibility, Deprecation, and Migration Plan

This KIP will not deal with changing retry backoff behavior for other Kafka clients such as Kafka Connect and Kafka Streams. However since Kafka Connect also utilizes `ConsumerNetworkClient` and `Metadata`, those aspects with respect to the Kafka Connect framework will be affected. However other places that utilize the `retry.backoff.ms` configuration will not be affected, notably the classes responsible for rebalancing. Therefore, it will not affect Connect worker or Consumer group rebalancing. This KIP intends to replace the old static retry backoff behavior in Kafka clients with a more dynamic retry backoff behavior. While the behavior is replaced, the configuration won't be. `retry.backoff.ms` will still be utilized in calculating the exponential retry backoff (as long as `retry.backoff.ms` < `retry.backoff.max.ms`) as detailed in the "Public Interfaces" section of this KIP.

Rejected Alternatives

1. Default `retry.backoff.max.ms` to the same value as `retry.backoff.ms` so that existing behavior is always maintained: If a user isn't aware of this feature and doesn't set this, they wouldn't enjoy the exponential backoff. It's important to ensure that we provide this as a default feature for all Kafka clients. Since we generally don't expect users to change these settings, it's important to ensure we give good defaults out of the box.
2. Default `retry.backoff.max.ms` to be 1000 ms unconditionally: While we generally don't expect users to change these settings, we still would like to be able to give the option for them to do so, and be able to tune these settings to what suits their Kafka environment.