KIP-19 - Add a request timeout to NetworkClient

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: KAFKA-2120

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

In old producer/consumer, we have a socket timeout associated with each request, the producer/consumer will send a request and then wait for response. If no response is received from a broker within specified timeout, the request will fail.

In the NetworkClient of new producer/consumer, currently we don't have a similar timeout for requests. Adding a client side request timeout in NetworkClient would be useful for the following reasons:

1. For KafkaProducer.close() and KafkaProducer.flush() we need the request timeout as implict timeout.

2. Currently the producer sends requests in a non-blocking way and does not wait for response. If a socket is disconnected the network client will fail the requests with a NetworkException. However, It might take very long for TCP connection to timeout. Adding a client side request timeout will help solve such kind of issues.

This KIP is only trying to solve the problem after a request has been constructed. For new producer specifically, there are situations where record batches can sit in accumulator for very long before they are drained out of RecordAccumulator (KAKFA-1788).

We want to achieve the following with this timeout adjustments:

1. Clear timeout configurations - Configurations should be intuitive and and do exactly what they mean.
2. Have bounded blocking time for send() - user should be able to have a bounded blocking time when they call send()
3. No message/batch/requests will be able to reside in producer without a configurable timeout - currently batches can sit in accumulator for long time (KAFKA-1788), requests can be in-flight until TCP timeout (KAFKA-2120). In this KIP, we want to see how can we make sure we expire messages/batches/requests.

Public Interfaces

We will expose the following timeout in KafkaProducer:

1. **max.block.ms** - the maximum time producer.send() and partitionsFor() will block.
    a. For send() it includes:
        i. metadata fetch time
        ii. buffer full block time
        iii. serialization time (customized serializer)
        iv. partitioning time (customized partitioner)
    b. For partitionsFor() it includes:
        i. metadata fetch time
2. **request.timeout.ms** - the maximum time to wait for the response of a message AFTER the batch is ready, including:
    a. actual network RTT
    b. server replication time

Some thing to clarify:

1. Request timeout.ms only starts counting down after a batch is ready.
2. Request timeout.ms does not include retries - each try has a full request.timeout.ms. The reason we did not include retry here is that if we have the request timeout include retries, then we should actually keep retry until timeout. That indicates we should remove retry config but this causes problem for users who don't want duplicates. Also it is a little bit strange to timeout before user specified retries are exhausted.
3. Request timeout will also be used when the batches in the accumulator that are ready but not drained due to metadata missing - we are reusing request timeout to timeout the batches in accumulator.

The benefits of this approach are:

1. Only two configurations needed
2. Meet all the motivations
3. Easy for user to understand and easy to implement (all the batches in the same request will have same timeout)
4. The per message timeout is easy to compute - linger.ms + (retries + 1) * request.timeout.ms.
5. Good configuration isolation between components. request timeout will be a network client configuration and can be passed in to accumulator.

NOTE: In all the configurations above, replication timeout has been taken off the list according to a separate discussion thread on mailing list. We are going to use request timeout in ProducerRequest.

We need to have the following changes:

1. Add the following new configuration to org.apache.kafka.clients.producer.ProducerConfig and org.apache.kafka.clients.consumer.ConsumerConfig:

```
public static final String REQUEST_TIMEOUT_MS_CONFIG = "request.timeout.ms";
private static final String REQUEST_TIMEOUT_MS_DOC = "The configuration controls the maximum amount of time the
producer will wait " +
                                                     "for the response of a request. If the response is not
received before the timeout " +
                                                     "elapses the producer will resend the request if necessary or
fail the request if " +
                                                     "retries are exhausted."
```

If a request is timeout and retries have not been exhausted, the request will be retried.

When a request is timeout and retries are exhausted, an org.apache.kafka.common.errors.TimeoutException will be put in the returned future of a request and callback will be fired.

2. Add max.enqueue.timeout.ms to org.apache.kafka.clients.producer.KafkaProducer to bound the send() block time.

```
public static final String MAX_BLOCK_MS_CONFIG = "max.block.ms"
private static final String MAX_BLOCK_MS_DOC = "The configuration controls how long {@link KafkaProducer#send()}
will block." +
                                                     "The send method can be blocked because of buffer full,
metadata not available, " +
                                                     "customized partitioner/serialzier."
```

3. Remove METADATA_FETCH_TIMEOUT_CONFIG, TIMEOUT_CONFIG, BLOCK_ON_BUFFER_FULL_CONFIG.

Proposed Changes

Because the NetworkClient works in an async way. We need to keep track of the send time for each request. So the implementation will the following:

1. associate a time stamp with each inflight requests and retry
2. The poll() will go through the inflight requests and expire those requests if necessary.
3. The timeout for poll() will need to consider the request timeout for inflight requests.
4. When drain the data out of accumulator, we also check whether some batches should be expired.

The default request timeout will be set to a reasonable value, say 60 seconds.

Actions after request timeout

When the request timeout has been reached, we do the following:

1. disconnect TCP connection.
2. Refresh metadata
3. Retry the request on the new TCP connection

In most cases, metadata refresh should be able to pick up the new leader if a broker is down. If a broker was not down but just slow, as long as request timeout is set to a reasonable value, we should not see dramatic increase in TCP connections when a broker was not down but just slow.

Compatibility, Deprecation, and Migration Plan

To remove these configurations, we will do the following;

1. In 0.8.2.2 or 0.8.3, we will :
    a. Keep all the deprecated configuration there.

    b. If user sets TIMEOUT_CONFIG, we override REQUEST_TIMEOUT_CONFIG and show a deprecation warning and advertise the new configuration.
    c. If user set METADATA_FETCH_TIMEOUT_CONFIG, we override MAX_BLOCK_MS_CONFIG with METADATA_FETCH_TIMEOUT_CONFIG and warn about the deprecation.
    d. if user set BLOCK_ON_BUFFER_FULL_CONFIG to true, we override the MAX_BLOCK_MS_CONFIG with Long.MAX_VALUE and warn about the deprecation.
    e. if the user sets BLOCK_ON_BUFFER_FULL_CONFIG to true and also METADATA_FETCH_TIMEOUT_CONFIG, we will not honor METADATA_FETCH_TIMEOUT_CONFIG.
    We will warn the users regarding the same. Also, we will be warning them to use the MAX_BLOCK_MS_CONFIG explicitly.
2. In 0.9, we will remove TIMEOUT_CONFIG, METADATA_FETCH_TIMEOUT_CONFIG and BLOCK_ON_BUFFER_FULL_CONFIG.

Rejected Alternatives

Option 1

1. **max.buffer.full.block.ms** - To replace block.on.buffer.full. The max time to block when buffer is full.
2. **metadata.fetch.timeout.ms** - reuse metadata timeout as batch.timeout.ms because it is essentially metadata not available.
3. **network.request.timeout.ms** - This timeout is used when producer sends request to brokers through TCP connections. It specifies how long the producer should wait for the response.

With the above approach, we can achieve the following.

- Send() will need metadata and buffer available, so we can have bounded blocking time for send() = (1) + (2), not considering customized partitioner and serializer
- The time after send() until response got received is generally bounded by linger.ms + (2) + (3), not taking retries into consideration.

The benefit of this option are:

- User can have finer tuning on the producer timeout.
- Configuration scope and usage are cleaner and more precise compared with option 2. (Not committing on per-message-timeout while actually shoehorning timeout of some messages to others)

The downside of this option are:

- Need to educate user about metadata and buffer.
- The request timeout is less explicit but will be affected by retries, linger.ms, retry backoff time, request timeout, etc.

Option 2

1. **max.enqueue.block.ms** - the maximum time producer.send() will block, including:
    a. metadata fetch time
    b. buffer full block time
    c. serialization time (customized serializer)
    d. partitioning time (customized partitioner)
2. **request.timeout.ms** - the maximum time to wait for the response of a message after message has been appended to the accumulator. Including:
    a. linger.ms
    b. actual network RTT
    c. server replication time
    d. retries

(1) can be alternatively exposed as by a new send() API but this might be misleading as user might think of it as the overall timeout for the message.

(2) is tricky to implement. We need to solve the following issues:

Issue 1: Once a message is in the accumulator, we lose per message control.

Possible solution: One implementation is always use the most latest expiry time of a message in the same batch as the batch expiry time. In this case, a message might be expired a little bit later, but will not be expired pre-maturely. The difference might be up to linger.ms. Similarly after batches are drained to a request. We use the latest expiry time of all batches in the same request as the expiry time of the request.

Issue 2: If there is retry, ideally request.timeout.ms should be at least linger.ms + replication.timeout.ms(probably we can remove it from user config) + retries * retry.backoff.ms. This sanity check might be difficult to explain to user.

Possible Solution: We only enforce request.timout.ms > linger.ms + some default replication.timeout.ms. We failed the batch either retries are exhausted or request.timeout.ms is reached, which ever come first.

The benefit of this option are:

- Strait forward to user (assuming user does not care about the a little bit belated expiring of messages).
- Has more explicit boundary for send() and how long a message will be sent or failed.

The downside of this option are:

- Less accurate expiry time - message can be timeout out between (2) and 2*(2) + (1).
- Might be a little bit difficult to tune for some user (e.g.. Some user might be willing to wait for buffer but not metadata)