KIP-3 - Mirror Maker Enhancement

Status

Current state: Accepted

Discussion thread: *here*

JIRA: KAFKA-1997

Released: 0.8.3

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

Motivation

The Mirror Maker has a potential data loss issue as explained below:

1. Mirror Maker consumer consume some messages and called producer.send(). The messages sit in producer accumulator and haven't been sent yet.

2. Mirror Maker consumer commits the offsets

3. Mirror Maker somehow dies with some messages left in the producer accumulator not yet successfully delivered.

4. When Mirror Maker restarts, all the messages that has been consumed and committed by consumer but not successfully delivered by producer yet will be lost.

The new mirror maker design is trying to solve the above issue as well as enhance the the mirror maker with the following features.

1.Finer control over the target partition.

2.Allow user do some simple process of messages with a wired in message handler.

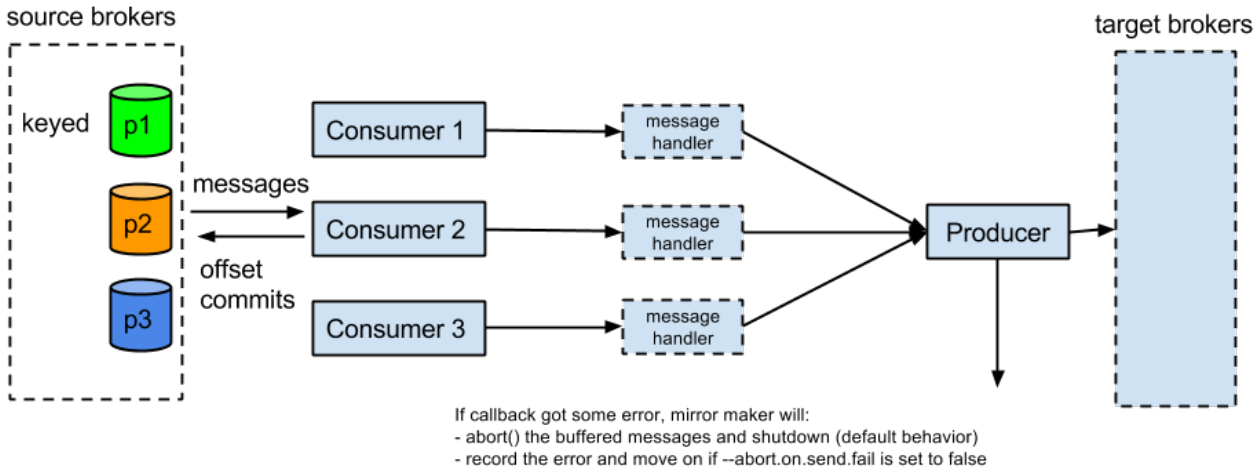
The new mirror maker is designed based on new consumer.

Public Interfaces

The following interface change will be made to mirror maker.

- **--consumer.rebalance.listener** - to allow user to wire in a custom rebalance listener which implements ConusmerRebalanceCallback interface. An internal rebalance listener which implements ConsumerRebalanceCallback is wired in by default to avoid duplicates on consumer rebalance. User could still specify a custom listener class in command line argument. The internal rebalance listener will call that custom listener after it finishes the default logic.
- **--consumer.rebalance.listener.args -** to provide arguments to custom rebalance listener constructor.
- **--message.handler** - to allow user do some simple process on the messages (e.g. filtering, reformatting, etc)
- **--message.handler.args** - to provide arguments for message handler init() function.
- **--abort.on.send.fail** - define whether mirror maker should abort when a send failed(turned on by default)

Proposed Changes



- Each mirror maker thread consume messages, process them and then send them.
- Scale by creating more mirror maker thread.

Each mirror maker thread has a consumer instance associated with it, the thread will be responsible for decompression, message handling, compression and offset commit. All mirror maker threads share a producer.

The consumer's offsets auto-commits should be turned off, and the offsets will be committed periodically following a flush() call on producer.

The mirror maker thread logic is as below.

```
while (!shutdown) {
        val records = consumer.poll(timeout)
        for (record <- records) {
                val handledRecords = messageHandler.handle(record)
                handledRecords.forEach(producer.send)
        }
    if (System.currentMilliSeconds - lastOffsetCommitMs > offsetCommitIntervalMs) {
                producer.flush()
                consumer.commitOffsets()
        }
}
```

Offset commit

Each mirror maker thread periodically flush() all the messages in producer and then commit offsets.

No data loss option

A no data loss option is provided with the following settings, and these are also **default settings**:

1. For consumer: auto.commit.enabled=false
2. For producer:
    a. max.in.flight.requests.per.connection=1
    b. retries=Int.MaxValue
    c. acks=-1
    d. block.on.buffer.full=true
3. set --abortOnSendFail

The following actions will be taken by mirror maker:

- Mirror maker will only send one request to a broker at any given point.
- If any exception is caught in mirror maker thread, mirror maker will try to commit the acked offsets then exit immediately.
- For RetriableException in producer, producer will retry indefinitely. If retry did not work, eventually the entire mirror maker will block on producer buffer full.
- For None-retriable exception, if --abort.on.send.fail is specified, stop the mirror maker. Otherwise producer callback will record the message that was not successfully sent but let the mirror maker move on. **In this case, that message will be lost in target cluster.**

On consumer rebalance

A new consumerRebalanceListener needs to be added to make sure there is no duplicates when consumer rebalance occurs.

```
/**
 * This listener is used for execution of tasks defined by user when a consumer rebalance
 * occurs in {@link kafka.consumer.ZookeeperConsumerConnector}
 */
public interface ConsumerRebalanceListener {
    /**
     * This method is called after all the fetcher threads are stopped but before the
     * ownership of partitions are released. Depending on whether auto offset commit is
     * enabled or not, offsets may or may not have been committed.
     * This listener is initially added to prevent duplicate messages on consumer rebalance
     * in mirror maker, where offset auto commit is disabled to prevent data loss. It could
     * also be used in more general cases.
     */
    public void beforeReleasingPartitions(Map<String, Set<Integer>> partitionOwnership);
  /**
   * This method is called after the new partition assignment is finished but before fetcher
   * threads start. A map of new global partition assignment is passed in as parameter.
   * @param consumerIdString The consumer instance invoking this rebalance callback
   * @param partitionAssignment The partition assignment result of current rebalance
   */
    public void beforeStartingFetchers(String consumerIdString, Map<String, Map<Integer, ConsumerThreadId>>
partitionAssignment);
}
```

The callback beforeReleasingPartitions will be invoked on consumer rebalance. It should

1. call producer.flush

2. commit the offsets

The default rebalance listener does not do anything in beforeStartingFetchers callback. This callback could be useful in some cases, for example, to create a topic in the target cluster with the same number of partitions as in source cluster.

Message handler in consumer thread

Sometimes it is useful for mirror maker to do some process on the messages it consumed such as filtering out some messages or format conversions. This could be done by adding a message handler in consumer thread. The interface could be

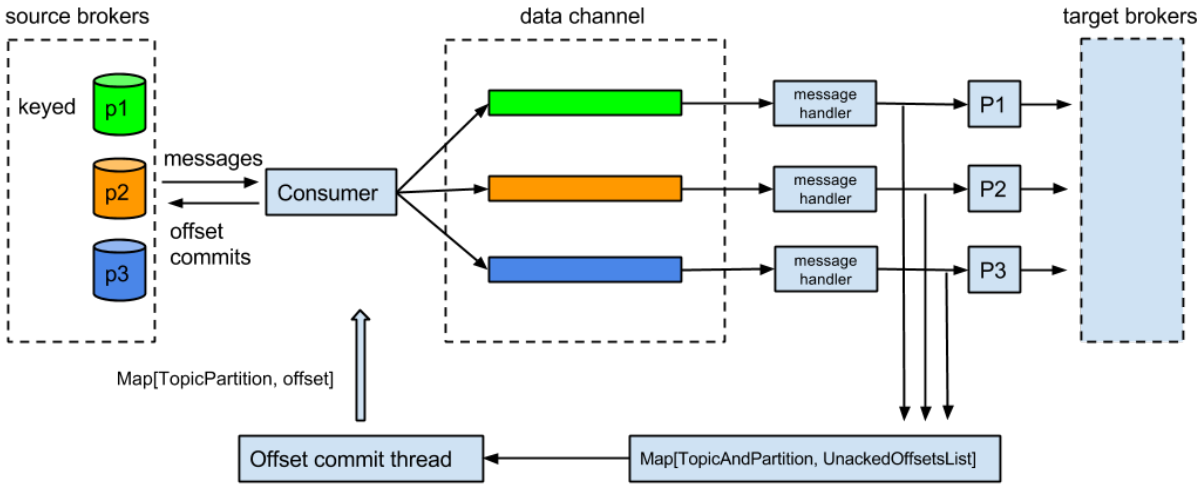
```
public interface MirrorMakerMessageHandler {
    /**
     * Initialize the custom message handler with passed in arguments. This function will be called when mirror
maker instantiate the custom message handler.
     */
    public void init(String args);
    /**
     * The message handler will be executed to handle messages in mirror maker
     * thread before the messages are handed to producer. The message handler should
     * not block and should handle all the exceptions by itself.
     */
    public List<ProducerRecord> handle(ConsumerRecord record);
}
```

The default message handler just return a list with passed in record.

Compatibility, Deprecation, and Migration Plan

The changes are backward compatible.

Rejected Alternatives


source brokers
data channel
target brokers
keyed
p1
p2
p3
messages
offset
commits
Consumer
message
handler
message
handler
message
handler
P1
P2
P3
Map[TopicPartition, offset]
Offset commit thread
Map[TopicAndPartition, UnackedOffsetsList]