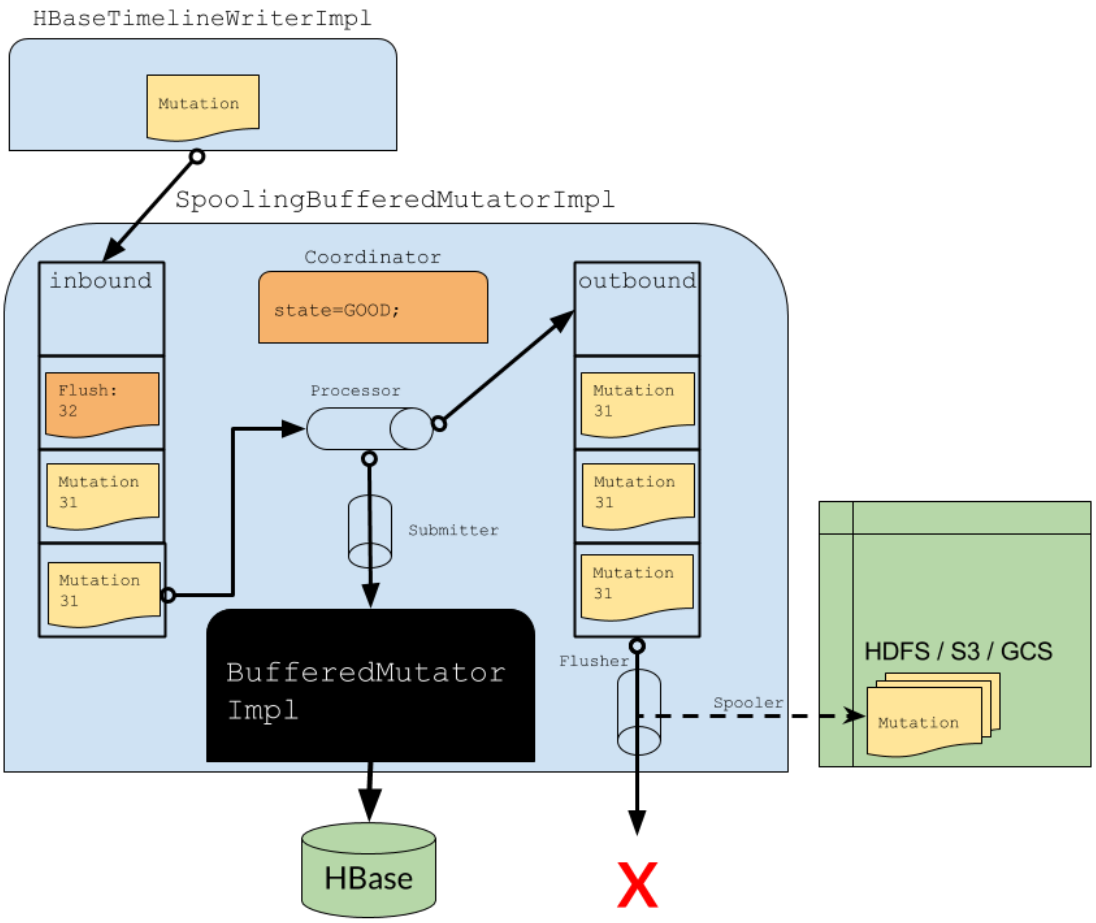
HBASE-17018 SpoolingBufferedMutator

SpoolingBufferedMutator

As part of previous discussions on HBASE-17018 we looked at modifying the BufferedMutatorImpl to allow access to its internal state. Due to the structure of the code this is a rather daunting undertaking.
Therefore, this design takes a different approach: it wraps an actual live BufferedMutatorImpl, and it treats as a black-box. The advantage is that we do not have to change its visibility from private and we'd be shielded from (most) internal changes. The downside is that we have to track additional state, because we don't know which data is buffered in the BufferedMutatorImpl unless we carefully keep track. We will keep an additional reference to mutations that are sent into the buffer.



HBaseTimelineWriterImpl

For the purposes of this design, this is simply a client submitting mutations through the `BufferedMutator`, and would flush (and close) as previously with the `BufferedMutatorImpl` implementation.

SpoolingBufferedMutatorImpl

This class implements `BufferedMutator` and wraps a `BufferedMutatorImpl`, which it gets from the `ClusterConnection` by manipulating the `BufferedMutatorParams` as per HBASE-17277.

There are two supporting queues, an inbound queue, and an outbound queue. `Mutation`s are wrapped in a `SpoolingBufferedMutatorSubmission` which comes in three

`SpoolingBufferedMutatorSubmissionType` s: CLOSE, FLUSH, MUTATE. The latter point to the mutation they wrap. Each is marked with a flushCount, which helps keep track of the "batch" of submissions that it is part of. The submissions are queued in the queues.

The `BufferedMutatorImpl` is the only entity that will enqueue items in the inbound queue. A `SpoolingBufferedMutatorProcessor` running on a separate thread is responsible to dequeue items from the inbound queue, hand them to a `SpoolingBufferedMutatorSubmitter` . A `BufferedMutatorSubmitter` runs on a separate thread and is responsible to submit to the wrapped `BufferedMutatorImpl`. The `BufferedMutatorProcessor` is the only entity dequeuing items from the inbound queue. Each time an items is pulled from the inbound queue, it is immediately queued in the outbound queue. The `BufferedMutatorProcessor` asks the `SpoolingufferedMutatorCoordinator` how long to wait for submissions to complete. The `SpoolingufferedMutatorCoordinator` is responsible to keep track of the HBase state.
A `SpoolingufferedMu tatorFlusher` is responsible for flushing items out of the outbound queue. If `SpoolingufferedMutatorCoordinator` tells it shouldSpool then the it will use a `SpoolingufferedMutatorSpooler` to spool items, otherwise items are known to be flushed by the `SpoolingufferedMutatorImpl` and items are dropped.

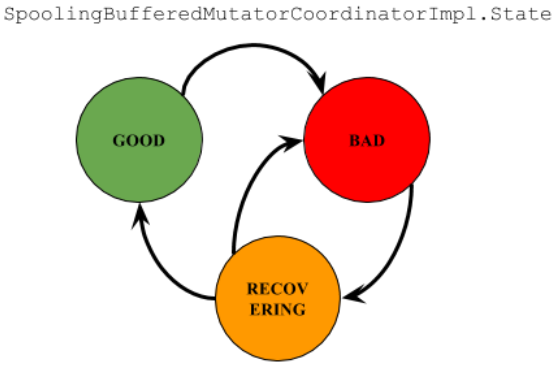
Both flush and close methods on `SpoolingufferedMutatorImpl` are synchronized and block until the operation completes. Each "batch" of mutations will be flushed from the outbound queue. In the flush() and close() methods of `SpoolingufferedMutatorImpl` a `SpoolingBufferedMutatorFlusher` is submitted to the flusher Executor Service. The flusher doesn't yet know whether all the items in the "flush batch" should be spooled or dropped. Only when the flush (and close) submission is either submitted or timedout, do we know what to do with the batch., The `SpoolingBufferedMutatorFlusher` will therefore wait on the Submission to be ready to be flushed (spooled or dropped). The processor sets this state on the submission.
This design assumes that `BufferedMutatorImpl` will (eventually) recover from HBase outages.

SpoolingBufferedMutatorProcessor

The processor runs on a separate thread (managed by an ExecutorService) and is responsible to block on the inbound queue until submissions arrive. It then hands submissions to a SpoolingBufferedMutatorSubmitter which runs on its own separate thread (because it can block). The `SpoolingBu fferedMutatorCoordinator` tells the processor how long to wait for submissions to complete. In the normal state, submissions will finish before timing out. Once the `SpoolingBufferedMutatorCoordinator` informs the processor to no longer wait for the submission (because it concluded that HBase is in a BAD state)

SpoolingBufferedMutatorCoordinator

The coordinator is responsible to keep track of the HBase state. It can be in three states:



In the good state, batches of mutations are dropped per flush. In both the bad and the recovering state they are spooled.
In the GOOD and RECOVERING states, the coordinator will instruct the processor to wait for submissions to complete. If another timeout or error occurs, then the state goes back to BAD. If enough good submissions and flushes go through the `BufferedMutatorImpl` , then the state will revert back to GOOD.
Mutations can therefore be both submitted to the `BufferedMutatorImpl` to HBase as well as spooled. The caller is responsible to either de-dupe spooled mutations, or ensure that they are idempotent.

SpoolingBufferedMutatorSubmitter

The submitter has the very simple task to submit an item to the wrapped `BufferedMutatorImpl` , whether it is a mutate, a flush, or a close submission. It runs on a separate thread so that when this submission hangs the processor can continue to inform the coordinator and eventually start spooling if HBase is determined to be in a bad state.

SpoolingBufferedMutatorFlusher

The flusher is responsible to take items out of the outbound queue and either drop them or flush them, depending on what the coordinator indicates. There will be one batch of flushes per flush call on the `SpoolingBufferedMutatorImpl` . Such a flush callable are enqueued in the flush ExecutorService, which will run only one of the flushers at a time. The flusher will block until the flush submission indicates it is ready. That is triggered by the coordinator when the flush to the wrapped `BufferedMutatorImpl` either succeeds or times out (indicating HBase is in a bad state). When a flush submission to HBase succeeds, then mutations in that batch are dropped from the outbound queue, otherwise they are spooled.
Since the mutate methods on the `SpoolingBufferedMutatorImpl` are not synchronized, the mutations can be enqueued in the inbound queue out of order with respect to the flush count. Therefore, the flusher will re-queue the items that aren't older than the current flush count.

MutationSpooler

This interface allows multiple implementations of spoolers to be used.

FilesystemMutationSpooler

The filesystem mutation spooler implements `MutationSpooler` and will take a configuration pointing to a filesystem path, which it will then use to spool files. The path can be an HDFS path, a GCS, or an s3 filesystem path. The format will likely be protobufs, possibly in WAL format, so that existing implementations to replay WALs can be used.