WAN Gateway Sender Callback API & Dead-letter queue example implementation

Overview

The WAN replication feature allows 2 remote data centers, or 2 availability zones, to maintain data consistency. In the case where one data center cannot process incoming events for any reason, the other data center should retain the failed events so that no data is lost. Currently if data center 1 (DC1) is able to connect to data center 2 (DC2) and send it events, those events are removed from the queue on DC1 when the ack from DC2 is received, regardless of what happens to them on DC2. This behavior is controlled by the internal system property REMOVE_FROM_QUEUE_ON_EXCEPTION which defaults to true. Most common exceptions thrown from a receiving site include:

- LowMemoryException - when one or more members is low on memory
- CacheWriterException - when a CacheWriter before* method throws an exception
- PartitionOfflineException - when all the members defining a persistent bucket are offline
- RegionDestroyedException - when the region doesn't exist
- Malformed data exception (unable to deserialize)

Goals

We will provide a mechanism for users to preserve events on the gateway sender that do not get successfully processed on the receiving data center. Our example implementation will store these events on disk at the sending data center and notify the user what events did not get transmitted.

1. Deprecate (and later remove) the internal system property REMOVE_FROM_QUEUE_ON_EXCEPTION, but detect if it is set to false and support existing behavior (infinite retries)
2. Create a new callback API that will be executed when an exception is returned with the acknowledgement from the receiver
3. Provide an example implementation of the callback that saves events with exceptions returned from the receiver in a 'dead-letter' queue on the sender (on disk)
4. Add a new property for the gateway sender to specify the path to the custom implementation of the callback.
   a. If no path is provided, use default, example implementation
   b. if property is not specified, revert to existing behavior (removing events from the queue when ack is received, ignoring batch exceptions)
5. Add 2 new properties for the gateway receiver to control when to send the acknowledgement with the exceptions:
   a. the number of retries for events failing with an exception
   b. the wait time between retries

Not in Scope

1. Providing the ability to directly replay events from the dead-letter queue.

Approach

Our current design approach is as follows:

1. Deprecate existing internal boolean system property: REMOVE_FROM_QUEUE_ON_EXCEPTION
   a. Continue to support default behavior if boolean set to false by setting # retries on receiver to -1
2. Create new Java API

   a. Define callback API for senders to set callback to dispatchers
   b. If sender is configured with a callback, invoke the callback if batch exception occurs prior to batch removal
   c. Implement a default callback API (see item 5 below)
   d. Add properties on gateway receiver factory to specify # retries for a failed event and wait time between retries.
3. Modify Gfsh commands

   a. Add option to gfsh ‘create gateway sender’ command to specify custom callback
   b. Add options to gfsh ‘create gateway receiver’ command to set # retries and wait time between retries

   c. Store new options in cluster config

      i. Sender: callback implementation
      ii. Receiver: # of retries and wait time between retries
4. Add support in cache.xml for specifying new callback for gateway sender and setting new options for gateway receiver
5. Create example implementation of Sender callback that writes event(s) and associated exceptions to a file
6. Security features

   a. Define privileges needed to deploy and configure sender callback
   b. With security, callback should only write eventIds and exceptions, i.e. no entry values should be written to disk.
7. Add logging and statistics for callback

   a. Log messages for gateway receiver for start time and results of retries
   b. Add statistics and MBean for callbacks in-progress, completed, # and duration

New workflow for setting up WAN gateway using gfsh:

1. Create gateway receiver including new options for specifying # of retries and wait time between retries
2. Deploy jar on gateway sender(s) containing callback implementation
3. Create gateway sender with option to add callback

New API + Changes to existing APIs

Java Definition

The new *GatewayEventFailureListener* interface is defined like:

GatewayEventFailureListener

```
public interface GatewayEventFailureListener extends CacheCallback {

  /**
   * Callback invoked on the GatewaySender when an event fails to be processed by the
   * GatewayReceiver
   *
   * @param event The event that failed
   *
   * @param exception The exception that occurred
   */
  void onFailure(GatewayQueueEvent event, Throwable exception);
}
```

Example:

LoggingGatewayEventFailureListener

```
public class LoggingGatewayEventFailureListener implements GatewayEventFailureListener, Declarable {

  private Cache cache;

  public void onFailure(GatewayQueueEvent event, Throwable exception) {
    this.cache.getLogger().warning("LoggingGatewayEventFailureListener onFailure: region=" + event.getRegion().
getName() + "; operation=" + event.getOperation() + "; key=" + event.getKey() + "; value=" + event.
getDeserializedValue() + "; exception=" + exception);
  }

  public void initialize(Cache cache, Properties properties) {
    this.cache = cache;
  }
}
```

This **LoggingGatewayEventFailureListener** will log warnings like:

```
[warning 2018/11/05 17:30:41.613 PST ln-1 <AckReaderThread for : Event Processor for GatewaySender_ny_3>
tid=0x75] LoggingGatewayEventFailureListener onFailure: region=data; operation=CREATE; key=8360; value=Trade
[id=8360; cusip=PVTL; shares=100; price=18]; exception=org.apache.geode.cache.persistence.
PartitionOfflineException: Region /data bucket 73 has persistent data that is no longer online stored at these
locations: [...]
```

Java Configuration

GatewaySender

The **GatewaySenderFactory** adds the ability to add a **GatewayEventFailureListener**:

```
/**
 * Sets the provided <code>GatewayEventFailureListener</code> in this GatewaySenderFactory.
 *
 * @param listener The <code>GatewayEventFailureListener</code>
 */
GatewaySenderFactory setGatewayEventFailureListener(GatewayEventFailureListener listener);
```

The **GatewaySender** adds the ability to get a **GatewayEventFailureListener**:

```
/**
 * Returns this <code>GatewaySender's</code> <code>GatewayEventFailureListener</code>.
 *
 * @return this <code>GatewaySender's</code> <code>GatewayEventFailureListener</code>
 */
GatewayEventFailureListener getGatewayEventFailureListener();
```

Example:

```
GatewaySender sender = cache.createGatewaySenderFactory()
        .setParallel(true)
        .setGatewayEventFailureListener(new FileGatewayEventFailureListener(new File(...)))
        .create("ln", 2);
```

GatewayReceiver

The **GatewayReceiverFactory** adds the ability to set retry attempts and wait time between retry attempts:

```
/**
 * Sets the number of retry attempts to apply failing events from remote GatewaySenders
 *
 * @param retryAttempts The retry attempts
 */
GatewayReceiverFactory setRetryAttempts(int retryAttempts);

/**
 * Sets the wait time between retry attempts to apply failing events from remote GatewaySenders
 *
 * @param waitTimeBetweenRetryAttempts The wait time in milliseconds
 */
GatewayReceiverFactory setWaitTimeBetweenRetryAttempts(long waitTimeBetweenRetryAttempts);
```

The **GatewayReceiver** adds the ability to get retry attempts and wait time between retry attempts:

```
/**
 * Returns the number of times to retry a failing event before throwing an exception.
 *
 * @return the number of times to retry a failing event before throwing an exception
 */
int getRetryAttempts();

/**
 * Returns the amount of time in milliseconds to wait between attempts to apply a failing event.
 *
 * @return the amount of time in milliseconds to wait between attempts to apply a failing event
 */
long getWaitTimeBetweenRetryAttempts();
```

Example:

```
GatewayReceiver receiver = cache.createGatewayReceiverFactory()
        .setRetryAttempts(10)
        .setWaitTimeBetweenRetryAttempts(100)
        .create();
```

Gfsh Configuration

gateway-sender

The **create gateway-sender** command defines this new parameter:

Name	Description
gateway-event-failure-listener	The fully qualified class name of GatewayEventFailureListener to be set in the GatewaySender

Example:

```
Cluster-1 gfsh>create gateway-sender --id=ln --parallel=true --remote-distributed-system-id=2 --gateway-event-
failure-listener=LoggingGatewayEventFailureListener
Member | Status
------ | -----------------------------------
ny-1   | GatewaySender "ln" created on "ny-1"
```

gateway-receiver

The **create gateway-receiver** command defines these new parameters:

Name	Description
retry-attempts	The number of retry attempts for failed events processed by the GatewayReceiver
wait-time-between-retry-attempts	The amount of time to wait between retry attempts for failed events processed by the GatewayReceiver

Example:

```
Cluster-2 gfsh>create gateway-receiver --retry-attempts=10 --wait-time-between-retry-attempts=100
Member | Status | Message
------ | ------ | ---------------------------------------------------------------------------
ln-1   | OK     | GatewayReceiver created on member "ln-1" and will listen on the port "5296"
```

XML Configuration

gateway-sender

The **<gateway-sender>** element defines the **<gateway-event-failure-listener>** sub-element. The **<gateway-event-failure-listener>** sub-element is like any other **Declarable**.

Example:

```
<gateway-sender id="...">
        <gateway-event-failure-listener>
                <class-name>FileGatewayEventFailureListener</class-name>
        </gateway-event-failure-listener>
</gateway-sender>
```

gateway-receiver

The **<gateway-receiver>** element defines the *retry-attempts* and *wait-time-between-retry-attempts* attributes.

Example:

```
<gateway-receiver retry-attempts="5" wait-time-between-retry-attempts="100"/>
```

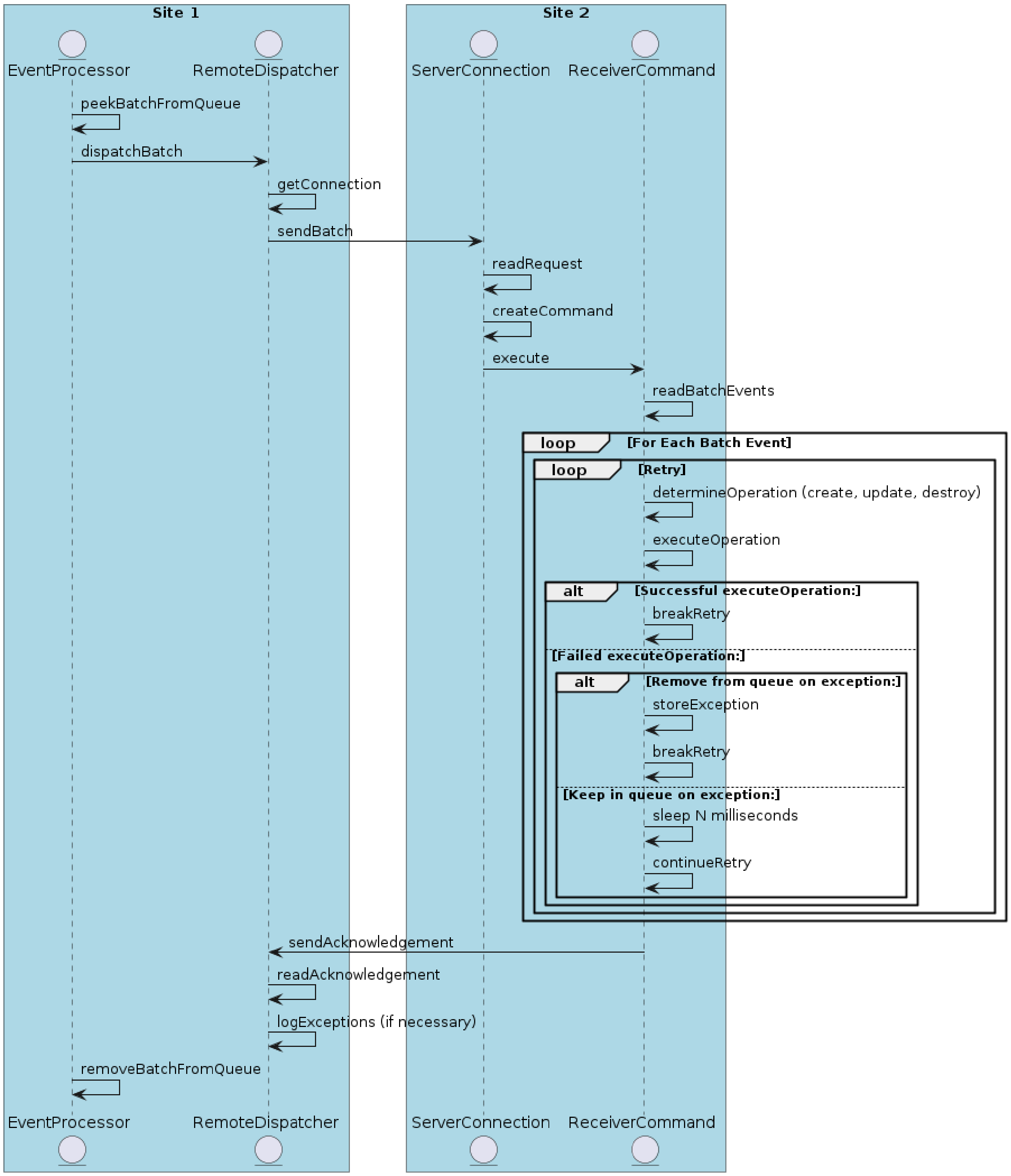
Risks and Unknowns

1. How to handle class not found exception for sender callback
2. Default behavior when no callback is provided for sender? - Should be same as current behavior
3. Backward compatibility behavior
    a. old sender connected to new receiver using new options
    b. new sender with callback implemented connected to old receiver
4. Sort out security privileges needed for deploying vs installing with sender vs reading values for failed events written to disk.

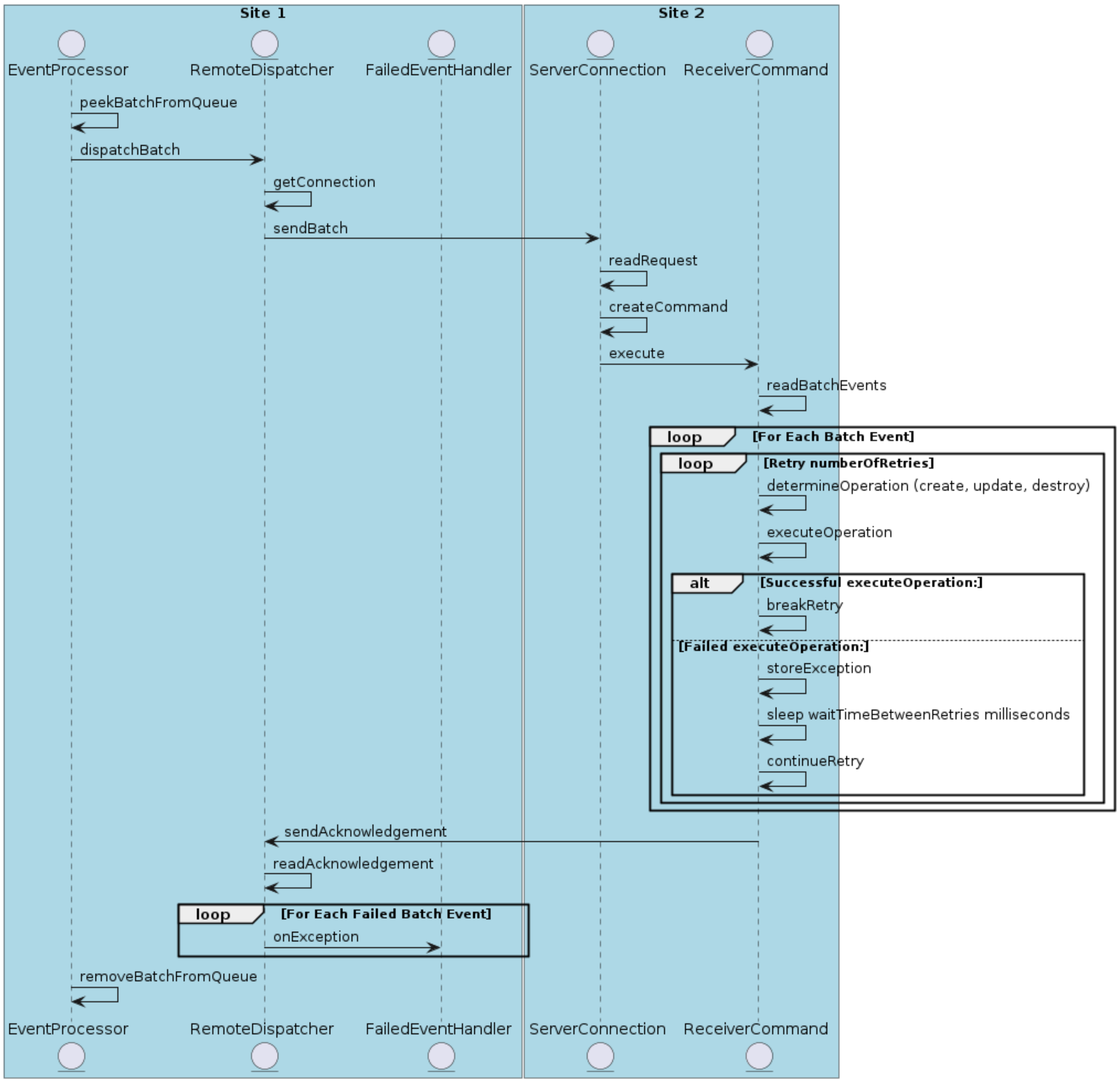
Potential Future Enhancements

- Ability to modify batch removal to remove specific events from the batch
- Ability to resend events saved in dead-letter queue

Current Implementation



Proposed Implementation


Site 1
Site 2
EventProcessor
RemoteDispatcher
FailedEventHandler
ServerConnection
ReceiverCommand
peekBatchFromQueue
dispatchBatch
getConnection
sendBatch
readRequest
createCommand
execute
readBatchEvents
loop [For Each Batch Event]
loop [Retry numberOfRetries]
determineOperation (create, update, destroy)
executeOperation
alt [Successful executeOperation:]
breakRetry
[Failed executeOperation:]
storeException
sleep waitTimeBetweenRetries milliseconds
continueRetry
sendAcknowledgement
readAcknowledgement
loop [For Each Failed Batch Event]
onException
removeBatchFromQueue