DBZ-175 Alternate Proposal

Motivation

The Debezium MySQL connector can be used to capture a snapshot of data in a MySQL server plus all subsequent inserts, updates, and deletes of rows within that server's tables. The connector requires no special columns or schema changes, but can instead do this with any and all tables, regardless of their schema. The connector taps into MySQL's replication mechanism, which involves the MySQL server recording the changes within its binlog that can be read by other MySQL servers that replicate those same changes. The Debezium MySQL connector is in effect another MySQL replica, except that the Debezium connector records the changes in Kafka topics, where they can be read and processed by any number of stream processing applications. As long as the MySQL connector runs, it will continue to capture the changes and write the to the Kafka topics.

When a new Debezium MySQL connector is deployed, the database filter and table filter dictate which of the tables in the MySQL server are to be captured. When the connector starts, it first performs an initial snapshot to capture the current state of each table whose fully-qualified name satisfies the configured filter. When the snapshot is complete, the connector then proceeds to read the MySQL binlog, processing only those events in the binlog that correspond to the same tables that were copied in the snapshot. The connector can detect when new tables are added, and it uses the filters and the table's fully qualified names to determine whether the table is to be captured or ignored.

Each of the resulting Kafka topics is a stream of change events that describe the initial state of all rows in the particular table followed by all changes to the rows in the same order they were made in the MySQL database. The stream can be used to replicate the complete contents of the table.

Once the connector has been running, however, it is not currently feasible to change the connector's filter to include existing tables that were previously excluded. Yes, one can easily stop the connector, change the filters in the configuration to include additional tables, and restart the connector. The connector begins reading the binlog where it previously left off, and it does include events for the newly added tables. However, the connector currently doesn't perform an initial snapshot of the newly added tables, which means the corresponding topics are missing the events that describe the rows that existed at the time the connector was restarted. In other words, the change data stream for the newly-added tables is incomplete and of far less value.

We would like to change the Debezium MySQL connector to handle this case by producing the complete change event streams for the additional tables. Of course, this requires performing an initial snapshot of the additional tables, and these may take some time to complete, especially when the additional tables have large numbers of rows. So the challenge becomes how to do this snapshotting while also continuing to process the binlog for the original set of tables.

Proposed Approach

The existing MySQL connector has several `Reader` components that encapsulate different logic. One of these is the `SnapshotReader` that knows how to perform a snapshot. Another is the `BinlogReader` that knows how to read the binlog. There is also a `ChainedReader` that is simply a list of other readers that are run sequentially. Each `Reader` can be given a function that it will call when it has completed its work.

Normally when the `MySqlConnectorTask` starts up for the first time it creates and then starts a `ChainedReader` with the following readers:

ChainedReader (SnapshotReader → BinlogReader)

where → denotes sequential execution. When the `MySqlConnectorTask` starts up and it determines that snapshot is not required, it creates and then starts a `ChainedReader` with just one `BinlogReader`:

ChainedReader (BinlogReader)

We will add a few other types of readers and make a few more changes to support the concurrent snapshot process, with the end result being the task can create the following `ChainedReader` when it determines that the filters have changed and it needs to perform a snapshot with the newly added tables:

ChainedReader (ParallelSnapshotReader → ReconcilingBinlogReader → BinlogReader):

Here, the `ParallelSnapshotReader` is actually a composite Reader that will simultaneously read the binlog as normal for the original set of tables *and* perform a snapshot of the newly-added tables followed by reading the binlog for just those tables. After the snapshot is complete, there are two threads reading the binlog: the first is reading the binlog and is theoretically close to the head of the binlog, and the second is reading the binlog from the point the snapshot was started. Eventually, both will be reading the binlog *near* the head of the binlog, and when they are they should both stop and the `ReconcilingBinlogReader` takes over. We cannot guarantee that both readers have stopped at the same point, so the purpose of the `ReconcilingBinlogReader` is to reconcile these two positions into a single position in the binlog. When this is complete, the normal binlog reader starts up and runs continuously.

The rest of this section outlines the steps required to get to this point.

Step 1: Ability to stop reading the binlog

The first change is to modify the `BinlogReader` to accept a `BinlogReader.Predicate` function that it can use to detect whether to keep reading the binlog given the most recently read binlog filename and position. By default, the `BinlogReader` will have a no-op function that always returns true to reflect the current functionality. Note that when the reader stops, it should invoke the reader's completion handler that was registered via the reader's `uponCompletion(Runnable)` method.

Step 2: Create BinlogReader.Predicate function to stop near a specified position in the binlog

The next change is to implement the `BinlogReader.Predicate` function that will return true as long as the supplied "current" filename/position is not within a fixed distance of a specified "stop" filename/position. This function should allow the "stop" filename/position to be set from a different thread (so using volatile).

Step 3: Create ReconcilingBinlogReader

This new class is a Reader that implements the two cases identified by WePay: either the table snapshot binlog position, A, is slightly ahead or behind that the binlog reader, B. So, this reader

needs to just read the binlog from A to B or B to A, filtering the events appropriately. This implementation can use a `BinlogReader` and a custom `BinlogReader.Predicate` function to stop the binlog reading at point A or B.

Step 4: Add ParallelSnapshotReader

This class is a Reader that will run a slightly more complex graph of readers. It will simultaneously start a normal `BinlogReader` to read the binlog with the original set of tables, and a `ChainedReader` composed of a `SnapshotReader` to perform the snapshot of the added tables followed by a `ReconcilingBinlogReader`.

ParallelSnapshotReader:
BinlogReader
ChainedReader (SnapshotReader → BinlogReader)

This reader will configure the snapshot branch's `BinlogReader` with a Predicate function that will stop it when it has (nearly) reached the `BinlogReader`'s current position (e.g., each time the `BinlogReader`'s current position is recorded, the Predicate function's "stop" position will be updated). Therefore, while the `BinlogReader` will continue until it is stopped, the `ChainedReader` will *eventually* stop once the snapshot has completed and the binlog has been read up to a point near where the `BinlogReader` should be.

Step 5: Update the MySqlConnectorTask

When the task determines that new tables have been added and need to be snapshot, it will create a `ChainedReader` with the following structure:

ChainedReader (ParallelSnapshotReader → ReconcilingBinlogReader → BinlogReader):

rather than just the current options of:

ChainedReader (SnapshotReader → BinlogReader)

or:

ChainedReader (BinlogReader)

Change 6: Change the offsets
These readers need to write offsets that are aware of the different states. TBD.

Rejected Alternatives

There were several alternative approaches that were initially considered but were ultimately rejected.

Snapshot All Tables

This approach works with the existing, unmodified connector. Each time additional tables are to be captured, the existing connector is stopped, its configuration is changed to include the additional tables and a new connector name, and the connector is then started. The new connector name means that the previously stored offsets will not be used, and so the connector begins a new snapshot of all of the tables.

The time required to perform the snapshot is a function of the total number of records in all of the tables. Therefore, this approach is relatively effective when the tables are small, but it becomes unattractive when the tables are larger. Another disadvantage is that snapshot is not required for the tables that had been generated before the configuration change, and this will result in excess and unnecessary writes to the corresponding topics.

Use Separate Connectors

Another approach that works with the existing, unmodified connector is to use a new connector to capture the additional tables. This may be acceptable once or twice, but it becomes unwieldy if this happens more frequently, as the number of connectors simply continues to grow.

Pause Reading the Binlog During Snapshot of Additional Tables

The approach required modifying the connector to recognize that additional tables were included in the filter, and to perform the initial snapshot of those tables before starting to read the binlog for the complete set of tables. While this is a relatively straightforward change, this approach had a number of significant disadvantages. First, it requires modifying some of the core (and complicated) snapshot logic. Second, because the snapshot would be performed at a different binlog position the where the connector last left off, when the snapshot is complete the

connector would first have to read the binlog with only the previous set of tables until it reached the new snapshot binlog position, at which point it would have to now include events for all of the tables. Finally, this results in the connector not reading the binlog and producing changes during the snapshot, which for large tables can be a significant amount of time.

Snapshot While Continuing to Read the Binlog

This approach is similar to the previous one, except that the connector continues to read the binlog while it is performing a snapshot. In fact, this is similar to the proposed approach, except that the implementation would involve changing the internals of the existing connector in a less desirable way than the proposed approach.

Use Separate Connect Tasks

This goal of this approach was to reuse as much of the existing connector codebase as possible, and to rely upon the Connect task mechanism to run the normal binlog reader task and the auxiliary snapshot+binlog reader task in parallel. Once the auxiliary task reaches the head of the binlog, it would request Connect perform a task reconfiguration, at which point the tasks are stopped and the connector is prompted to compute new task configuration. If the Connector were able to read the offsets (see https://issues.apache.org/jira/browse/KAFKA-4794), then the Connector could detect that the auxiliary task completed and only a single task need to be started. (The single task does need to reconcile the slightly different binlog positions, but this would be a simple short-lived step that could be accomplished with a “ReconciliationReader”.) However, that is not currently possible with Apache Kafka 1.0.

Use Pseudo Connect Tasks

This approach is similar to the previous approach, except that since the Connect API doesn’t let the Connector read the offsets and determine the tasks, we instead have to do this ourselves. This would involve creating an uber SourceTask implementation that knows whether to run one or two of the tasks, and how to reconcile them. The advantage of this approach is that it would lend itself to migrating to letting Connect manage the tasks once KAFKA-4794 has been completed. However, it’s still not quite as simple as the proposed approach, and would require changes down the road to have Connect manage the tasks.