# Introduction

As part of the work to add ACID INSERT, UPDATE, and DELETE to Hive (see HIVE-5317) a compactor is needed. This compactor will be responsible for both minor and major compactions of tables and partitions. This document lays out the requirements and design for this compactor.

# Definitions

**Base File**: A traditional HDFS file holding data for a Hive table or partition.

**Delta File**: A file storing inserted, updated, or deleted records.

**Major Compaction**: A compaction in which one or more delta files is combined with the base file. As part of this operation old versions of rows (deleted rows, updated rows) will be removed. Also, the metastore's transaction table will remove aborted transactions that referenced the table or partition being compacted. The old delta files will be removed once all readers are finished reading them.

**Minor Compaction**: A compaction that combines multiple delta files together. The base file will not be changed. The transaction table in the metastore will not be changed. The old, smaller delta files will be removed once all readers are finished reading them.

# Use Cases

1. User initiated compaction. A user may wish to initiate compaction on a table or partition that he owns. This could be a minor or major compaction. Compacting a table is only valid if the table is not partitioned.
2. System initiated compaction. The system will monitor partitions, and for non-partitioned tables tables, to see if they need minor or major compactions. Based on configurable criteria (such as number of delta files, size of the delta relative to the size of the base, number of aborted transactions against the partition) the system will decide to initiate a compaction.

# Requirements

Based on the use cases given above, the compactor will meet the following requirements:

1. Compaction can be initiated on a table (non-partitioned only) or a partition by the owner of the table. Attempts by user A to initiate compaction on a table owned by user B will be rejected (except in the case where system A is the system, see below).
    a. User initiation of compaction will be done from SQL via a command like `COMPACT TABLE tablename [partition (partkey = partval, …)]`.
2. The system can initiate compaction on a partition or a non-partitioned table when the partition or table exceeds certain parameters (see below).

    a. If the underlying directories and files are owned by the user, this compaction will be done as the user who owns the table.
        i. In secure mode, the system will need to have credentials to impersonate the owner in order to compact tables.
    b. If the underlying directories and files are owned by the user running the metastore, then the compaction will be done as the use running the metastore.
3. The system will constrain the number of compactions it initiates simultaneously to avoid using excessive resources in compaction.
    a. The number of simultaneous compactions allowed will be configurable.
4. When a user compacts a partition or a non-partitioned table, compaction will be done regardless of whether the criteria for compaction is met, unless there is no work to do at all.
5. The table owner can turn off automated (system initiated) compaction for a table by setting a table property.
6. Minor compaction will be initiated based on the number of delta files in a partition or non-partitioned table.
    a. The number of delta files at which minor compaction is initiated will be configurable.
7. Major compaction will be initiated when either of the following hold:
    a. The delta file(s) is/are a certain percentage of the size of the base file.
    b. A given number of aborted transactions in the metastore refer to the partition or table to be compacted.
8. Compaction will be cognizant of the bucketing and sorting of a table and preserve that bucketing and sorting when compacting.

# Design

## Changes Elsewhere in Hive

A few changes will be required in existing Hive components to enable the compactor.

### Changes to Transaction Database

The transaction database that records transaction information will need to change to track transaction components after a transaction has closed. Currently the table `TXN_COMPONENTS` tracks all database, table, and partition name of any table or partition involved in a transaction. This is used to determine when an aborted transaction can be removed from the `TXNS` table and no longer reported as open to clients. This table will need to change to keep records for committed transactions between when the transaction is committed and when the referenced partition or table is compacted.

### Changes to Parser and DDLAnalyzer

Changes will be made to the parser to accept the new `COMPACT` statement defined above. When this statement is executed the request will be placed on the queue used by the Initiator (see below).

### Changes to Thrift Interface

A new call will be added to the Metastore Thrift interface to transmit the compaction request from the client to the metastore.

## Initiator

The initiator will be responsible for determining partitions or tables that are ready for compaction. It will scan the `TXN_COMPONENTS` table every `hive.compactor.check.interval` seconds. This value will be set fairly high (5 minutes by default) as each check will involve many operations on HDFS. Also, it is assumed that once compactions are initiated it will take some time for the worker threads to finish them.

For a given table, if the table parameter `NO_AUTO_COMPACTION` is set then no automated compaction will be done on that table. Users will still be able to manually initiate compaction.

A major compaction will be run if `hive.compactor.abortedtxn.threshold` is exceeded and `NO_AUTO_COMPACTION` is not set. A major compaction will also be initiated if the sum of the sizes of the delta files exceeds `hive.compactor.delta.pct.threshold`. Major compactions will be checked for before minor compactions. This avoids the issue of doing a minor compaction on a partition and then immediately following it with a major compaction.

Any partition or non-partitioned table that has at least `hive.compactor.delta.num.threshold` entries in the `TXN_COMPONENTS` table that are committed will be examined to see if a minor compaction should be run. A minor compaction will be run if at least `hive.compactor.delta.num.threshold` delta files are found and `NO_AUTO_COMPACTION` is not set.

The initiator will not execute any of the compactions. It will place all transactions that should be done in a queue to be picked up by a worker thread. This queue will be stored in the metastore's RDBMS so that it is durable across metastore failures. All entries in the queue will start in the `initiated` state.

In addition to initiating compactions the initiator will look for failed compactions. If the initiator finds any entries in the queue that are in the `in-progress` state and being worked on by a worker thread in the same metastore and that thread is no longer alive, then the entry will be placed back in `initiated` state. If it finds any entries in the queue in the `in-progress` state that are being worked on by a different metastore and they are more than one day old, then they will be placed back in `initiated` state.

The initiator will run in a thread in the metastore. It will only be started if `hive.compactor.initiator.on` is set to true.

## Worker

Workers will do the actual work of compaction. The number of worker threads started will be determined by `hive.compactor.worker.threads`. They will pull work from the queue filled by the initiator. They will mark the entry in the queue as `in-progress`, place a unique identifier on the record noting which thread is working on it, and the date at which the work began.

Before beginning a compaction the worker thread will determine whether the table or partition to be compacted is bucketed and/or sorted. If so, it will preserve these traits during compaction.

In the case where the underlying directories and files are owned by the owner of the table (and not the user running the metastore) the worker thread will need to execute a doAs() to switch to impersonate the owner of the data. The MR jobs used to do compaction will be run as the owner.

Compaction will be achieved by map only jobs. There will be separate jobs written for minor and major compaction (and likely for sorted versus non-sorted). Sorting can be done in the map because each map will be handling a single bucket.

In the minor compaction case the MR job will simply consist of reading all of the input files and writing a single output file.

In major compaction the MR job will read the input file(s), keep only the most recent version of each record, and then write a single file.

Once a worker is done compacting a partition or table, it will place the entry in the queue in `waiting-for-cleaning` state.

If the MR job fails, the worker thread will place the queue entry back in `initiated` state and return to the queue to look for new entries to work on.

## Cleaner

The cleaner thread will clean up after compaction. It will be a separate thread. It will be started only if `hive.compactor.initiator.on` is set to true. Once every minute it will:

1. Look in the queue for any records in `waiting-for-cleaning` state. For any it finds it will request an exclusive lock on that partition or table.
2. Check existing locks to see if they have been obtained.

Once a lock is obtained the cleaner will remove all files that are not the most recent version. This can be determined by the transaction ids in the directory names. For a discussion of directory naming conventions in tables using ACID, see the ACID design document. In the case of major compaction it will also initiate a cleaning of the `TXN_COMPONENTS` table and `TXNS` table

for the partition or table being compacted. This will remove all references to the partition or table being compacted from `TXN_COMPONENTS`. For each transaction that has a row removed from `TXN_COMPONENTS`, if there are no other entries in `TXN_COMPONENTS` for that transaction (that is, no other partitions or tables participated in the transaction) then the associated transaction record will be removed from `TXNS`.

## Failure Semantics

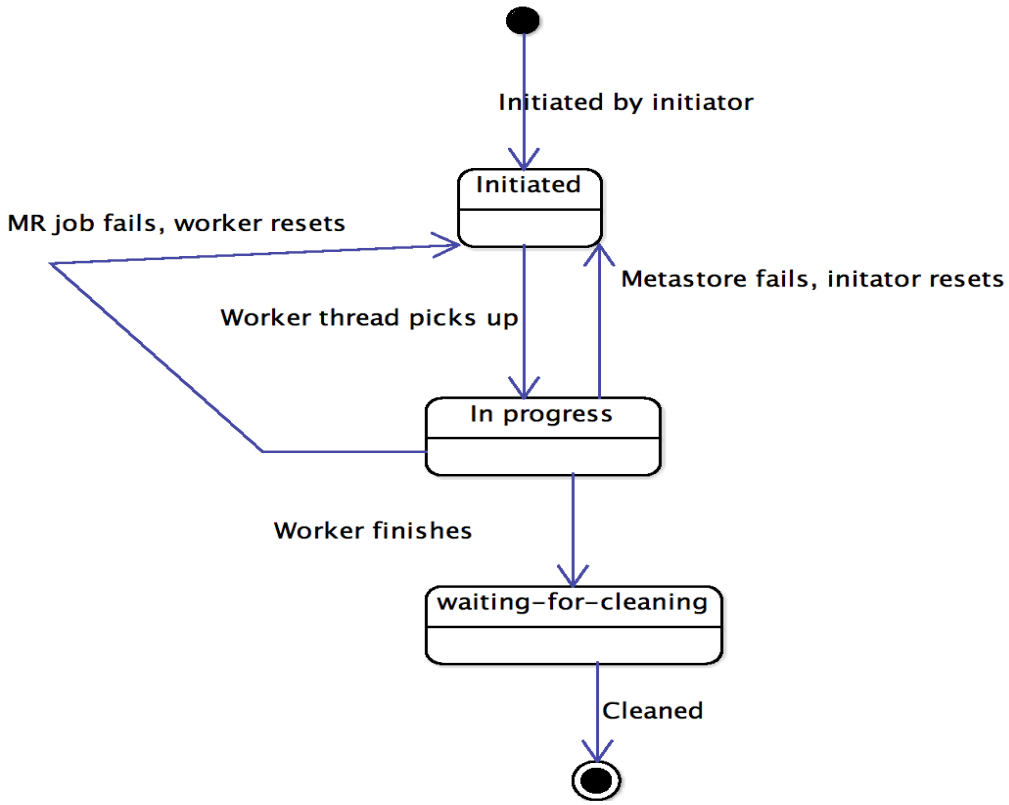
The compactor must handle the case where there is a failure at any point during compaction. These failures could come from Hadoop failures while running the MR jobs. They could also come from a metastore failure, or the user shutting down the metastore.

In the event of metastore failure the initiator, worker, and cleaner threads will be restarted when the metastore is restarted. No monitor thread will exist to assure these threads are running.

| **Failure** | **State** | **Recovery Mechanism** |
|---|---|---|
| Metastore | `initiated` | None required. |
| Metastore | `in-progress` | Initiator will detect loss of worker thread on restart. If in separate metastore initiator will detect after one day. |
| MR job | `in-progress` | Worker will detect and put job back in `initiated` state. |
| Metastore | `waiting-for-cleaning`, lock not yet requested | None required. |
| Metastore | `waiting-for-cleaning`, lock requested | Lock will expire from lack of heartbeat, new cleaner thread will pick up reccord. |

## State Transition Diagram

The following diagram covers the state transitions for the entries in the compactor queue.




## New Configurable Parameters

| Parameter | Description | Default |
|---|---|---|

| | | |
|---|---|---|
| `hive.compactor.check.interval` | Frequency with which initiator checks `TXN_COMPONENTS` table to look for partitions/tables to be compacted, in seconds. | 300 |
| `hive.compactor.delta.num.threshold` | Number of delta files in a table or partition that will cause minor compaction to be initiated. | 10 |
| `hive.compactor.initiator.on` | Whether the initiator thread is started at metastore start up. | false |
| `hive.compactor.delta.pct.threshold` | Size of the delta file(s) relative to the base at which major compaction will be initiated, in percentage. | 10 |
| `hive.compactor.abortedtxn.threshold` | Number of aborted transactions relating to a given table or partition after which major compaction will be done on that table or partition. | 1000 |
| `hive.compactor.worker.threads` | Number of worker threads to do compactions. | 0 |