Hive Transactions

Created by Alan Gates, last modified by Eugene Koifman on Mar 30, 2016

ACID and Transactions in Hive

What is ACID and why should you use it?

ACID stands for four traits of database transactions: Atomicity (an operation either succeeds completely or fails, it does not leave partial data), Consistency (once an application performs an operation the results of that operation are visible to it in every subsequent operation), Isolation (operations by one user do not cause unexpected side effects for other users), and Durability (once an operation is complete it will be preserved even in the face of machine or system failure). These traits have long been expected of database systems as part of their transaction functionality.

Up until Hive 0.13, atomicity, consistency, and durability were provided at the partition level. Isolation could be provided by turning on one of the available locking mechanisms (ZooKeeper or in memory). With the addition of transactions in Hive 0.13 it is now possible to provide full ACID semantics at the row level, so that one application can add rows while another reads from the same partition without interfering with each other.

Transactions with ACID semantics have been added to Hive to address the following use cases:

1. Streaming ingest of data. Many users have tools such as Apache Flume, Apache Storm, or Apache Kafka that they use to stream data into their Hadoop cluster. While these tools can write data at rates of hundreds or more rows per second, Hive can only add partitions every fifteen minutes to an hour. Adding partitions more often leads quickly to an overwhelming number of partitions in the table. These tools could stream data into existing partitions, but this would cause readers to get dirty reads (that is, they would see data written after they had started their queries) and leave many small files in their directories that would put pressure on the NameNode. With this new functionality this use case will be supported while allowing readers to get a consistent view of the data and avoiding too many files.
2. Slow changing dimensions. In a typical star schema data warehouse, dimensions tables change slowly over time. For example, a retailer will open new stores, which need to be added to the stores table, or an existing store may change its square footage or some other tracked characteristic. These changes lead to inserts of individual records or updates of records (depending on the strategy chosen). Starting with 0.14, Hive is able to support this.
3. Data restatement. Sometimes collected data is found to be incorrect and needs correction. Or the first instance of the data may be an approximation (90% of servers reporting) with the full data provided later. Or business rules may require that certain transactions be restated due to subsequent transactions (e.g., after making a purchase a customer may purchase a membership and thus be entitled to discount prices, including on the previous purchase). Or a user may be contractually required to remove their customer's data upon termination of their relationship. Starting with Hive 0.14 these use cases can be supported via *INSERT, UPDATE*, and *DELETE*.

Limitations

- *BEGIN*, *COMMIT*, and *ROLLBACK* are not yet supported. All language operations are auto-commit. The plan is to support these in a future release.
- Only ORC file format is supported in this first release. The feature has been built such that transactions can be used by any storage format that can determine how updates or deletes apply to base records (basically, that has an explicit or implicit row id), but so far the integration work has only been done for ORC.
- By default transactions are configured to be off. See the Configuration section below for a discussion of which values need to be set to configure it.
- Tables must be bucketed to make use of these features. Tables in the same system not using transactions and ACID do not need to be bucketed.
- At this time only snapshot level isolation is supported. When a given query starts it will be provided with a consistent snapshot of the data. There is no support for dirty read, read committed, repeatable read, or serializable. With the introduction of BEGIN the intention is to support snapshot isolation for the duration of transaction rather than just a single query. Other isolation levels may be added depending on user requests.
- The existing ZooKeeper and in-memory lock managers are not compatible with transactions. There is no intention to address this issue. See Basic Design below for a discussion of how locks are stored for transactions.
- Schema changes using ALTER TABLE is NOT supported for ACID tables. HIVE-11421 is tracking it.
- Using Oracle as the Metastore DB and "datanucleus.connectionPoolingType=BONECP" may generate intermittent "No such lock.." and "No such transaction..." errors. Setting "datanucleus.connectionPoolingType=DBCP" is recommended in this case.

Streaming APIs

Hive offers APIs for streaming data ingest and streaming mutation:

- Hive HCatalog Streaming API
- HCatalog Streaming Mutation API (available in Hive 2.0.0 and later)

A comparison of these two APIs is available in the Background section of the Streaming Mutation document.

Grammar Changes

INSERT...VALUES, UPDATE, and *DELETE* have been added to the SQL grammar, starting in Hive 0.14. See LanguageManual DML for details.

Several new commands have been added to Hive's DDL in support of ACID and transactions, plus some existing DDL has been modified.

A new command *SHOW TRANSACTIONS* has been added, see Show Transactions for details.

A new command *SHOW COMPACTIONS* has been added, see Show Compactions for details.

The *SHOW LOCKS* command has been altered to provide information about the new locks associated with transactions. If you are using the ZooKeeper or in-memory lock managers you will notice no difference in the output of this command. See Show Locks for details.

A new option has been added to *ALTER TABLE* to request a compaction of a table or partition. In general users do not need to request compactions, as the system will detect the need for them and initiate the compaction. However, if compaction is turned off for a table or a user wants to compact the table at a time the system would not choose to, *ALTER TABLE* can be used to initiate the compaction. See Alter Table/Partition Compact for details. This will enqueue a request for compaction and return. To watch the progress of the compaction the user can use *SHOW COMPACTIONS*.

Basic Design

HDFS does not support in-place changes to files. It also does not offer read consistency in the face of writers appending to files being read by a user. In order to provide these features on top of HDFS we have followed the standard approach used in other data warehousing tools. Data for the table or partition is stored in a set of base files. New records, updates, and deletes are stored in delta files. A new set of delta files is created for each transaction (or in the case of streaming agents such as Flume or Storm, each batch of transactions) that alters a table or partition. At read time the reader merges the base and delta files, applying any updates and deletes as it reads.

Delta File Compaction

Occasionally these changes need to be merged into the base files. To do this a set of threads has been added to the Hive metastore. They determine when this compaction needs to be done, execute the compaction, and then clean up afterwards. There are two types of compactions, minor and major.

- Minor compaction takes a set of existing delta files and rewrites them to a single delta file per bucket.
- Major compaction takes one or more delta files and the base file for the bucket and rewrites them into a new base file per bucket.

All compactions are done in the background and do not prevent concurrent reads and writes of the data. After a compaction the system waits until all readers of the old files have finished and then removes the old files.

Compactions are MR jobs with name in the following form: <hostname>-compactor-<db>.<table>.<partition>

Base and Delta Directories

Previously all files for a partition (or a table if the table is not partitioned) lived in a single directory. With these changes, any partitions (or tables) written with an ACID aware writer will have a directory for the base files and a directory for each set of delta files.

Lock Manager

A new lock manager has also been added to Hive, the DbLockManager. This lock manager stores all lock information in the metastore. In addition all transactions are stored in the metastore. This means that transactions and locks are durable in the face of server failure. To avoid clients dying and leaving transaction or locks dangling, a heartbeat is sent from lock holders and transaction initiators to the metastore on a regular basis. If a heartbeat is not received in the configured amount of time, the lock or transaction will be aborted.

As of Hive 1.3.0, the length of time that the DbLockManger will continue to try to acquire locks can be controlled via hive.lock.numretires and hive.lock.sleep.between.retries. When the DbLockManager cannot acquire a lock (due to existence of a competing lock), it will back off and try again after a certain time period. In order to support short running queries and not overwhelm the metastore at the same time, the DbLockManager will double the wait time after each retry. The initial back off time is 100ms and is capped by hive.lock.sleep.between.retries. hive.lock.numretries is the total number of times it will retry a given lock request. Thus the total time that the call to acquire locks will block (given default values of 10 retries and 60s sleep time) is (100ms + 200ms + 400ms + ... + 51200ms + 60s + 60s + ... + 60s) = 91m:42s:300ms.

More details on locks used by this Lock Manager.

Configuration

These configuration parameters must be set appropriately to turn on transaction support in Hive:

- hive.support.concurrency – true
- hive.enforce.bucketing – true (Not required as of Hive 2.0)
- hive.exec.dynamic.partition.mode – nonstrict
- hive.txn.manager – org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
- hive.compactor.initiator.on – true (for exactly one instance of the Thrift metastore service)
- hive.compactor.worker.threads – a positive number on at least one instance of the Thrift metastore service

The following sections list all of the configuration parameters that affect Hive transactions and compaction. Also see Limitations above and Table Properties below.

New Configuration Parameters for Transactions

A number of new configuration parameters have been added to the system to support transactions.

Configuration key	Values	Location	Notes
hive.txn.manager	*Default:* org.apache.hadoop.hive.ql.lockmgr.DummyTxnManager *Value required for transactions:* org.apache.hadoop.hive.ql.lockmgr.DbTxnManager	Client/ HiveServer2	DummyTxnManager replicates pre Hive-0.13 behavior and provides no transactions.
hive.txn.timeout	*Default:* 300	Client/ HiveServer2/ Metastore	Time after which transactions are declared aborted if the client has not sent a heartbeat, in seconds. It's critical that this property has the same value for all components/services.[5]
hive.timedout.txn.reaper.start	*Default:* 100s	Metastore	Time delay of first reaper (the process which aborts timed-out transactions) run after the metastore starts (as of Hive 1.3.0).
hive.timedout.txn.reaper.interval	*Default:* 180s	Metastore	Time interval describing how often the reaper (the process which aborts timed-out transactions) runs (as of Hive 1.3.0).
hive.txn.max.open.batch	*Default:* 1000	Client	Maximum number of transactions that can be fetched in one call to

			open_txns().[1]
hive.compactor.initiator.on	*Default:* false *Value required for transactions:* true (for exactly one instance of the Thrift metastore service)	Metastore	Whether to run the initiator and cleaner threads on this metastore instance. It's critical that this is enabled on exactly one metastore service instance (not enforced yet). As of Hive 1.3.0 this property may be enabled on any number of metastore instances.
hive.compactor.worker.threads	*Default:* 0 *Value required for transactions:* > 0 on at least one instance of the Thrift metastore service	Metastore	How many compactor worker threads to run on this metastore instance.[2]
hive.compactor.worker.timeout	*Default:* 86400	Metastore	Time in seconds after which a compaction job will be declared failed and the compaction re-queued.
hive.compactor.cleaner.run.interval	*Default*: 5000	Metastore	Time in milliseconds between runs of the cleaner thread. (Hive 0.14.0 and later.)
hive.compactor.check.interval	*Default:* 300	Metastore	Time in seconds between checks to see if any tables or partitions need to be compacted.[3]
hive.compactor.delta.num.threshold	*Default:* 10	Metastore	Number of delta directories in a table or partition that will trigger a minor compaction.
hive.compactor.delta.pct.threshold	*Default:* 0.1	Metastore	Percentage (fractional) size of the delta files relative to the base that will trigger a major compaction. 1 = 100%, so the default 0.1 = 10%.
hive.compactor.abortedtxn.threshold	*Default:* 1000	Metastore	Number of aborted transactions involving a given table or partition that will trigger a major compaction.
hive.compactor.max.num.delta	Default: 500	Metastore	Maximum number of delta files that the compactor will attempt to handle in a single job (as of Hive 1.3.0).[4]
hive.compactor.job.queue	*Default*: "" (empty string)	Metastore	Used to specify name of Hadoop queue to which Compaction jobs will be submitted. Set to empty string to let Hadoop choose the queue (as of Hive 1.3.0).

[1]hive.txn.max.open.batch controls how many transactions streaming agents such as Flume or Storm open simultaneously. The streaming agent then writes that number of entries into a single file (per Flume agent or Storm bolt). Thus increasing this value decreases the number of delta files created by streaming agents. But it also increases the number of open transactions that Hive has to track at any given time, which may negatively affect read performance.

[2]Worker threads spawn MapReduce jobs to do compactions. They do not do the compactions themselves. Increasing the number of worker threads will decrease the time it takes tables or partitions to be compacted once they are determined to need compaction. It will also increase the background load on the Hadoop cluster as more MapReduce jobs will be running in the background.

[3]Decreasing this value will reduce the time it takes for compaction to be started for a table or partition that requires compaction. However, checking if compaction is needed requires several calls to the NameNode for each table or partition that has had a transaction done on it since the last major compaction. So decreasing this value will increase the load on the NameNode.

[4]If the compactor detects a very high number of delta files, it will first run several partial minor compactions (currently sequentially) and then perform the compaction actually requested.

[5]If the value is not the same active transactions may be determined to be "timed out" and consequently Aborted. This will result in errors like "No such transaction...", "No such lock ..."

Configuration Values to Set for *INSERT, UPDATE, DELETE*

In addition to the new parameters listed above, some existing parameters need to be set to support *INSERT ... VALUES, UPDATE,* and *DELETE*.

Configuration key	Must be set to
hive.support.concurrency	true (default is false)
hive.enforce.bucketing	true (default is false) (Not required as of Hive 2.0)
hive.exec.dynamic.partition.mode	nonstrict (default is strict)

Configuration Values to Set for Compaction

If the data in your system is not owned by the Hive user (i.e., the user that the Hive metastore runs as), then Hive will need permission to run as the user who owns the data in order to perform compactions. If you have already set up HiveServer2 to impersonate users, then the only additional work to do is assure that Hive has the right to impersonate users from the host running the Hive metastore. This is done by adding the hostname to `hadoop.proxyuser.hive.hosts` in Hadoop's `core-site.xml` file. If you have not already done this, then you will need to configure Hive to act as a proxy user. This requires you to set up keytabs for the user running the Hive metastore and add `hadoop.proxyuser.hive.hosts` and `hadoop.proxyuser.hive.groups` to Hadoop's `core-site.xml` file. See the Hadoop documentation on secure mode for your version of Hadoop (e.g., for Hadoop 2.5.1 it is at Hadoop in Secure Mode).

Table Properties

If a table is to be used in ACID writes (insert, update, delete) then the table property "`transactional=true`" must be set on that table, starting with Hive 0.14.0. Also, hive.txn.manager must be set to org.apache.hadoop.hive.ql.lockmgr.DbTxnManager either in hive-site.xml or in the beginning of the session before any query is run. Without those, inserts will be done in the old style; updates and deletes will be prohibited. However, this does not apply to Hive 0.13.0.

If a table owner does not wish the system to automatically determine when to compact, then the table property "`NO_AUTO_COMPACTION`" can be set. This will prevent all automatic compactions. Manual compactions can still be done with Alter Table/Partition Compact statements.

Table properties are set with the TBLPROPERTIES clause when a table is created or altered, as described in the Create Table and Alter Table Properties sections of Hive Data Definition Language. The "`transactional`" and "`NO_AUTO_COMPACTION`" table properties are case-sensitive in Hive releases 0.x and 1.0, but they are case-insensitive starting with release 1.1.0 (HIVE-8308).

No labels