WritePathForUsers

Cassandra Write Path

This section provides an overview of the Cassandra Write Path for developers who use Cassandra. Cassandra developers, who work on the Cassandra source code, should refer to the Architecture Internals developer documentation for a more detailed overview.

CassandraWritePath.png|alt=Cassandra Write Path|width=800 title=Cassandra Write Path|width=800!

The Local Coordinator

The local coordinator receives the write request from the client and performs the following:

1. Firstly, the local coordinator determines which nodes are responsible for storing the data:
    - The first replica is chosen based on hashing the primary key using the Partitioner; Murmur3Partitioner is the default.
    - Other replicas are chosen based on the replication strategy defined for the keyspace. In a production cluster this is most likely the NetworkTopologyStrategy.
2. The local coordinator determines whether the write request would modify an associated materialized view.

If write request modifies materialized view

When using materialized views it's important to ensure that the base table and materialized view are consistent, i.e. all changes applied to the base table MUST be applied to the materialized view. Cassandra uses a two-stage batch log process for this:

- one batch log on the local coordinator ensuring that an update is made on the base table to a Quorum of replica nodes
- one batch log on each replica node ensuring the update is made to the corresponding materialized view.

The process on the local coordinator looks as follows:

1. Create batch log. To ensure consistency, the batch log ensures that changes are applied to a Quorum of replica nodes, regardless of the consistently level of the write request. Acknowledgement to the client is still based on the write request consistency level.
2. The write request is then sent to all replica nodes simultaneously.

If write request does not modify materialized view

1. The write request is then sent to all replica nodes simultaneously.

In both cases the total number of nodes receiving the write request is determined by the replication factor for the keyspace.

Replica Nodes

Replica nodes receive the write request from the local coordinator and perform the following:

1. Write data to the Commit Log. This is a sequential, memory-mapped log file, on disk, that can be used to rebuild MemTables if a crash occurs before the MemTable is flushed to disk.
2. Write data to the MemTable. MemTables are mutable, in-memory tables that are read/write. Each physical table on each replica node has an associated MemTable.
3. If the write request is a DELETE operation (whether a delete of a column or a row), a tombstone marker is written to the Commit Log and MemTable to indicate the delete.
4. If row caching is used, invalidate the cache for that row. Row cache is populated on read only, so it must be invalidated when data for that row is written.
5. Acknowledge the write request back to the local coordinator.

The local coordinator waits for the appropriate number of acknowledgements from the replica nodes (dependent on the consistency level for this write request) before acknowledging back to the client.

If write request modifies materialized view

Keeping a materialized view in sync with its base table adds more complexity to the write path and also incurs performance overheads on the replica node in the form of read-before-write, locks and batch logs.

1. The replica node acquires a lock on the partition, to ensure that write requests are serialised and applied to base table and materialized views in order.
2. The replica node reads the partition data and constructs the set of deltas to be applied to the materialized view. One insert/update/delete to the base table may result in one or more inserts/updates/deletes in the associated materialized view.
3. Write data to the Commit Log.
4. Create batch log containing updates to the materialized view. The batch log ensures the set of updates to the materialized view is atomic, and is part of the mechanism that ensures base table and materialized view are kept consistent.
5. Store the batch log containing the materialized view updates on the local replica node.
6. Send materialized view updates asynchronously to the materialized view replica (note, the materialized view partition could be stored on the same or a different replica node to the base table).
7. Write data to the MemTable.
8. The materialized view replica node will apply the update and return an acknowledgement to the base table replica node.

9. The same process takes place on each replica node that stores the data for the partition key.

Flushing MemTables

MemTables are flushed to disk based on various factors, some of which include:

- commitlog_total_space_in_mb is exceeded
- memtable_total_space_in_mb is exceeded
- 'Nodetool flush' command is executed
- Etc.

Each flush of a MemTable results in one new, immutable SSTable on disk. After the flush, an SSTable (Sorted String Table) is read-only. As with the write to the Commit Log, the write to the SSTable data file is a sequential write operation. An SSTable consists of multiple files, including the following:

- Bloom Filter
- Index
- Compression File (optional)
- Statistics File
- Data File
- Summary
- TOC.txt

Each MemTable flush executes the following steps:

1. Sort the MemTable columns by row key
2. Write the Bloom Filter
3. Write the Index
4. Serialise and write the data to the SSTable Data File
5. Write the Compression File (if compression is used)
6. Write the Statistics File
7. Purge the written data from the Commit Log

Unavailable Replica Nodes and Hinted Handoff

When a local coordinator is unable to send data to a replica node due to the replica node being unavailable, the local coordinator stores the data either in its local system.hints table (prior to Cassandra v3.0) or in a local flat file (from Cassandra v3.0 onwards); this process is known as Hinted Handoff and is configured in cassandra.yaml. Hint data is stored for a default period of 3 hours, configurable using the max_hint_window_in_ms property in cassandra.yaml. If the replica node comes back online within the hinted handoff window the local coordinator will send the data to the replica node, otherwise the hint data is discarded and the replica node will need to be repaired.

Write Path Advantages

- The write path is one of Cassandra's key strengths: for each write request one sequential disk write plus one in-memory write occur, both of which are extremely fast.
- During a write operation, Cassandra never reads before writing (with the exception of Counters and Materialized Views), never rewrites data, never deletes data and never performs random I/O.