MemtableSSTable

Overview

Cassandra writes are first written to the CommitLog, and then to a per-ColumnFamily structure called a Memtable. When a Memtable is full, it is written to disk as an SSTable.

A Memtable is basically a write-back cache of data rows that can be looked up by key – that is, unlike a write-through cache, writes are batched up in the Memtable until it is full, when it is flushed.

Flushing

The process of turning a Memtable into a SSTable is called flushing. You can manually trigger flush via jmx (e.g. with bin/nodetool), which you may want to do before restarting nodes since it will reduce CommitLog replay time. Memtables are sorted by key and then written out sequentially. Thus, writes are extremely fast, costing only a commitlog append and an amortized sequential write for the flush.

Once flushed, SSTable files are immutable; no further writes may be done. So, on the read path, the server must (potentially, although it uses tricks like bloom filters to avoid doing so unnecessarily) combine row fragments from all the SSTables on disk, as well as any unflushed Memtables, to produce the requested data.

Compaction

To bound the number of SSTable files that must be consulted on reads, and to reclaim space taken by unused data, Cassandra performs compactions: merging multiple old SSTable files into a single new one. Compaction strategies are pluggable; out of the box are provided SizeTieredCompactionStrategy, which combines sstables of similar sizes, and LeveledCompactionStrategy, which sorts sstables into a heirarchy of levels, each an order of magnitude larger than the previous. As a rule of thumb, SizeTiered is better for write-intensive workloads, and Leveled better for read-intensive.

(For those familiar with other LSM implementations, it's worth noting that Cassandra can remove tombstones without a "major" compaction combining all sstables into a single file.)

Since the input SSTables are all sorted by key (technically, by token), merging in a compaction can be done efficiently, again requiring no random i/o. Even so, compaction can be a fairly heavyweight operation. Cassandra takes two steps to mitigate compaction impact on application requests:

1. Cassandra throttles compaction i/o to compaction_throughput_mb_per_sec (default 16MB/s)
2. Cassandra will request the operating system pull newly compacted partitions into its page cache when Cassandra's key cache indicates that the compacted partition was "hot" for recent reads

Once compaction is finished, the old SSTable files will be deleted as soon as any pending reads finish with them as well.

ColumnFamilyStoreMBean exposes sstable space used as getLiveDiskSpaceUsed (only includes size of non-obsolete files) and getTotalDiskSpaceUsed (includes everything), as well as statistics on average and max partition size.

Further reading

(The high-level memtable/sstable design as well as the "Memtable" and "SSTable" names come from Cassandra's sections 5.3 and 5.4 of Google's Bigtable paper, although some of the terminology around compaction differs.)

http://wiki.apache.org/cassandra/ArchitectureSSTable