Log-Structured Merge Trees (LSMs)

14-848 (Cloud Infrastrcuture)

Scenario

- A system is needed to support high-throughput updates
- The total data volume is larger than the main memory budget
- Writes to secondary storage occur more quickly and efficiently when batched than when written individually.
  - For example, writing a whole block of data at a time amortizes disk seek and rotational delay
- Sorting and indexing data in main memory can be done relatively efficiently causing relatively little delay.

Collect and Batch Updates In Memory

- Collect updates in memory
  - Sort them somehow
    - Sorted string list
    - Tree, Etc.
- Updates possible to in-memory values
  - But, once a value is written to disk, it stays written
  - Queries will need to find all records and merge
  - Tombstone deletes

Spill From Memory To Disk

- As memory budget approaches full, spill them to disk
  - Write out entire sorted string table
  - Write out a subtree, then remove and prune it in memory
- Each dump from memory to disk forms a “run” of some kind
  - Runs are time ordered

Merge, Idea #1

- Possibility #1:
  - Merge portions of in-memory data structure into on-disk data structure as spilled
  - Common when pruning in-memory trees and merging into on-disk trees
  - Slows the freeing of memory

Merge Idea #2

- Possibility #2:
  - Dump from memory into new “run”, i.e. data structure in secondary storage
  - Maintain in-memory Bloom Filters, one per disk run, to support queries
  - Upon query, check Bloom Filters
  - Then check on-disk runs only where Bloom filters indicated possible match
- Merge updates disk data structures in background
  - By similar tree pruning, if tree
  - By merging files into new files if tables
  - Delete then update Bloom Filter, since false positives aren’t fatal

Merge Idea #3

- Compaction occurs as part of the merges
  - Deletes Tombstoned records
  - Merges multiple updates into one
  - Recovers storage from merged updates and deleted values

Log-Structured Merge Trees (LSMs)

- When we spill subtrees or branches from an in-memory tree into a tree in secondary storage, this strategy is known as a Log-Structured Merge Tree (LSM) Tree
  - The in-memory tree is often known as C_0 and the tree in secondary storage is known as C_1.
  - If there are more levels of trees (not within a tree), they are known as C_1, C_2, etc.

Memtable, SSIndex, SSTable

- Common idiom in practice
  - Memtable in main memory contains sorted values and likely sorted <key, offset> index.
  - When spilled to disk is divided into SSTables and SSIndexes written separately
  - Indexes or Bloom Filters kept in memory
  - Merging in background when threshold met in terms of number of tables, etc.
  - Merges perform compaction
  - Write-Ahead logs used to aid recovery.
- Used in some form by Cassandra, Hbase, LevelDB, BigTable, Etc.

Summary

- Overall strategy
  - Fill memory
  - Spill to disk
  - Search disk runs until they can be merged
  - Use Bloom Filters to minimize unproductive searches
  - Updated in-memory, but merge independent changes once on disk.
- The overall strategy is sound even if it…
  - Does not involve trees, for example by using sorted string tables, and
  - Even if it leaves a forest of data structures to be searched after consulting a Bloom Filter.