Terminology

Edit New Page

Jump to bottom

Maysam Yabandeh edited this page on Dec 19, 2017 · 10 revisions

Iterator: iterators are used by users to query keys in a range in sorted order. See https://github.com/facebook/rocksdb/wiki/Basic-Operations#iteration

Point lookup: In RocksDB, point lookup means reading one key using Get().

Range lookup: Range lookup means reading a range of keys using an Iterator.

SST File (**Data file** / **SST table**): SST stands for Sorted Sequence Table. They are persistent files storing data. In the file keys are usually organized in sorted order so that a key or iterating position can be identified through a binary search.

Index The index on the data blocks in a SST file. It is persisted as an index block in the SST file. The default index format is the binary search index.

Partitioned Index The binary search index block partitioned to multiple smaller blocks. See https://github.com/facebook/rocksdb/wiki/Partitioned-Index-Filters

LSM-tree: See the definition in https://en.wikipedia.org/wiki/Log-structured_merge-tree RocksDB is LSM-tree-based storage engine.

Write-Ahead-Log (WAL) or **log**: A log file used to recover data that is not yet flushed to SST files, during DB recovery. See https://github.com/facebook/rocksdb/wiki/Write-Ahead-Log-File-Format

memtable / **write buffer**: the in-memory data structure that stores the most recent updates of the database. Usually it is organized in sorted order and includes a binary searchable index. See https://github.com/facebook/rocksdb/wiki/Basic-Operations#memtable-and-table-factories

memtable switch: During this process, the current **active memtable** (the one current writes go to) is closed and turned into an **immutable memtable**. At the same time, we will close the current WAL file and start a new one.

immutable memtable: A closed **memtable** that is waiting to be flushed.

sequence number (SeqNum / Seqno): each write to the database will be assigned an auto-incremented ID number. The number is attached with the key-value pair in WAL file, memtable, and SST files. The sequence number is used to implement snapshot read, garbage collection in compactions, MVCC in transactions, and some other purposes.

recovery: the process of restarting a database after it failed or was closed.

flush: background jobs that write out data in mem tables into SST files.

compaction: background jobs that merge some SST files into some other SST files. LevelDB's compaction also includes flush. In RocksDB, we further distinguished the two. See https://github.com/facebook/rocksdb/wiki/RocksDB-Basics#multi-threaded-compactions

Leveled Compaction or **Level-Based Compaction Style**: the default compaction style of RocksDB.

Universal Compaction Style: an alternative compaction algorithm. See https://github.com/facebook/rocksdb/wiki/Universal-Compaction

Comparator: A plug-in class which can define the order of keys. See https://github.com/facebook/rocksdb/blob/master/include/rocksdb/comparator.h

Column Family: column family is a separate key space in one DB. In spite of the misleading name, it has nothing to do with the "column family" concept in other storage systems. RocksDB doesn't even have the concept of "column". See https://github.com/facebook/rocksdb/wiki/Column-Families

snapshot: a snapshot is a logical consistent point-in-time view, in a running DB. See https://github.com/facebook/rocksdb/wiki/RocksDB-Basics#gets-iterators-and-snapshots

checkpoint: A checkpoint is a physical mirror of the database in another directory in the file system. See https://github.com/facebook/rocksdb/wiki/Checkpoints

backup: RocksDB has a backup tool to help users backup the DB state to a different location, like HDFS. See https://github.com/facebook/rocksdb/wiki/How-to-backup-RocksDB%3F

Version: An internal concept of RocksDB. A version consists of all the live SST files in one point of the time. Once a flush or compaction finishes, a new "version" will be created because the list of live SST files has changed. An old "version" can continue being used by on-going read requests or compaction jobs. Old versions will eventually be garbage collected.

Super Version: An internal concept of RocksDB. A super version consists the list of SST files (a "version") and the list of live mem tables in one point of the time. Either a compaction or flush, or a mem table switch will cause a new "super version" to be created. An old "super version" can continue being used by on-going read requests. Old super versions will eventually be garbage collected after it is not needed anymore.

block cache: in-memory data structure that cache the hot data blocks from the SST files. See See https://github.com/facebook/rocksdb/wiki/Block-Cache

Statistics: an in-memory data structure that contains cumulative stats of live databases. https://github.com/facebook/rocksdb/wiki/Statistics

perf context: an in-memory data structure to measure thread-local stats. It is usually used to measure per-query stats. See https://github.com/facebook/rocksdb/wiki/Perf-Context-and-IO-Stats-Context

DB properties: some running status can be returned by the function DB::GetProperty().

Table Properties: metadata stored in each SST file. It includes system properties that are generated by RocksDB and user defined table properties calculated by user defined call-backs. See https://github.com/facebook/rocksdb/blob/master/include/rocksdb/table_properties.h

write stall: When flush or compaction is backlogged, RocksDB may actively slowdown writes to make sure flush and compaction can catch up. See https://github.com/facebook/rocksdb/wiki/Write-Stalls

bloom filter: See https://github.com/facebook/rocksdb/wiki/RocksDB-Bloom-Filter

prefix bloom filter: a special bloom filter that can be limitly used in iterators. Some file reads are avoided if an SST file or memtable doesn't contain the prefix of the lookup key extracted by the prefix extractor. See https://github.com/facebook/rocksdb/wiki/Prefix-Seek-API-Changes

prefix extractor: a callback class that can extract prefix part of a key. This is most frequently used as the prefix used in prefix bloom filter. See https://github.com/facebook/rocksdb/blob/master/include/rocksdb/slice_transform.h

block-based bloom filter or **full bloom filter**: Two different approaches of storing bloom filters in SST files. see https://github.com/facebook/rocksdb/wiki/RocksDB-Bloom-Filter#new-bloom-filter-format

Partitioned Filters: Partitioning a full bloom filter into multiple smaller blocks. See https://github.com/facebook/rocksdb/wiki/Partitioned-Index-Filters.

compaction filter: a user plug-in that can modify or drop existing keys during a compaction. See https://github.com/facebook/rocksdb/blob/master/include/rocksdb/compaction_filter.h

merge operator: RocksDB supports a special operator Merge(), which is a delta record, merge operand, to the existing value. Merge operator is a user defined call-back class which can merge the merge operands. See https://github.com/facebook/rocksdb/wiki/Merge-Operator-Implementation

Block-Based Table: The default SST file format. See https://github.com/facebook/rocksdb/wiki/Rocksdb-BlockBasedTable-Format

Block: data block of SST files. In SST files, a block is stored in a compressed form.

PlainTable: An alternative format of SST file format, optimized for ramfs. See https://github.com/facebook/rocksdb/wiki/PlainTable-Format

forward iterator / tailing iterator: A special iterator option that optimizes for very specific use cases. See https://github.com/facebook/rocksdb/wiki/Tailing-Iterator

single delete: a special delete operation which only works when users never update an existing key: https://github.com/facebook/rocksdb/wiki/Single-Delete

rate limiter: it is used to limit the rate of bytes written to the file system by flush and compaction. See https://github.com/facebook/rocksdb/wiki/Rate-Limiter

Pessimistic Transactions Using locks to provide isolation between multiple concurrent transactions. The default write policy is WriteCommitted.

2PC (Two-phase commit) The pessimistic transactions could commit in two phases: first Prepare and then the actual Commit. See https://github.com/facebook/rocksdb/wiki/Two-Phase-Commit-Implementation

WriteCommitted The default write policy in pessimistic transactions, which buffers the writes in memory and write them into the DB upon commit of the transaction.

WritePrepared A write policy in pessimistic transactions that buffers the writes in memory and write them into the DB upon prepare if it is a 2PC transaction or commit otherwise. See https://github.com/facebook/rocksdb/wiki/WritePrepared-Transactions

WriteUnprepared A write policy in pessimistic transactions that avoid the need for larger memory buffers by writing data to the DB as they are sent by the transaction. See https://github.com/facebook/rocksdb/wiki/WritePrepared-Transactions

Documentation license here.

▸ Pages 106

Contents

    - Compaction
        - Leveled Compaction
        - Universal compaction style
        - FIFO compaction style
        - Manual Compaction
        - Sub-Compaction
    - Managing Disk Space Utilization
    - SST File Formats
        - Block-based Table Format
        - PlainTable Format
        - Bloom Filter
        - Data Block Hash Index
    - Logging and Monitoring
        - Logger
        - Statistics
        - Perf Context and IO Stats Context
        - EventListener
- **Tools / Utilities**
    - Administration and Data Access Tool
    - Benchmarking Tools
    - How to Backup RocksDB?
    - Checkpoints
    - How to persist in-memory RocksDB database
    - Stress Test
    - Third-party language bindings
    - RocksDB Trace, Replay, and Analyzer
- **Implementation Details**
    - Delete Stale Files
    - Partitioned Index/Filters
    - WritePrepared-Transactions
    - WriteUnprepared-Transactions
    - How we keep track of live SST files
    - How we index SST
    - Merge Operator Implementation
    - Choose Level Compaction Files
    - RocksDB Repairer
    - Two Phase Commit
    - Iterator's Implementation
    - Simulation Cache
    - Persistent Read Cache
- **RocksJava**
    - RocksJava Basics
    - RocksJava Performance on Flash Storage
    - JNI Debugging
    - Rocks Java API TODO

- RocksJava API TODO
- Lua
  - Lua CompactionFilter
- Performance
  - Performance on Flash Storage
  - In Memory Workload Performance
  - Read-Modify-Write Performance
  - Delete A Range Of Keys
  - Write Stalls
  - Pipelined Write
  - Tuning Guide
  - Memory usage in RocksDB
  - Speed-Up DB Open
  - Implement Queue Service Using RocksDB
- Misc
  - Building on Windows
  - Open Projects
  - Talks
  - Publication
  - Features Not in LevelDB
  - How to ask a performance-related question?
  - Articles about Rocks

Clone this wiki locally

```
https://github.com/facebook/rocksdb.wiki.git
```