

LevelDB Study

Introduction 2

2022. 07. 05

Presented by Min-guk Choi

koreachoi96@gmail.com

Previous Homework

- SSH

- Linux
 - Server
 - VM
- Window
 - Putty
 - Xshell
 - VS code

- VS code Extension

- Remote Development
 - <https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>
- C/C++ Extension Pack
 - <https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools-extension-pack>
- More Extension...
 - <https://lazyren.github.io/devlog/recommended-vscode-extension-list.html> (KOR)
 - <https://jhnyang.tistory.com/409> (KOR)

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- LevelDB Architecture
 - ✓ LSM-tree
 - Log-structured, Merge, Tree
 - ✓ LevelDB Implementation
 - Memtable, WAL, SSTable

LSM-tree

- What is LSM-tree
 - By Patrick O’Neil, The Log-Structured Merge Tree, 1996
 - Write optimized data structure
 - Log-structure
 - In-place update
 - ✓ good for read, bad for write
 - ✓ due to random writes
 - Out-of-place update
 - ✓ good for write, possible bad for read
 - ✓ due to multiple locations
 - ✓ need reclaiming mechanism

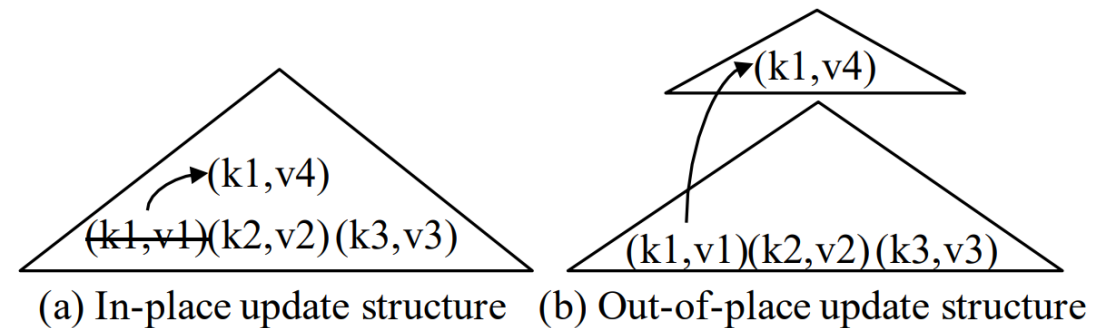


Fig. 1: Examples of in-place and out-of-place update structures: each entry contains a key (denoted as “k”) and a value (denoted as “v”)

(LSM-based Storage Techniques, VLDB Journal’19)

LSM-tree

- What is LSM-tree
 - Tree
 - larger at lower levels like a tree
 - C_0 is in main memory while $C_1 \sim C_K$ in Storage

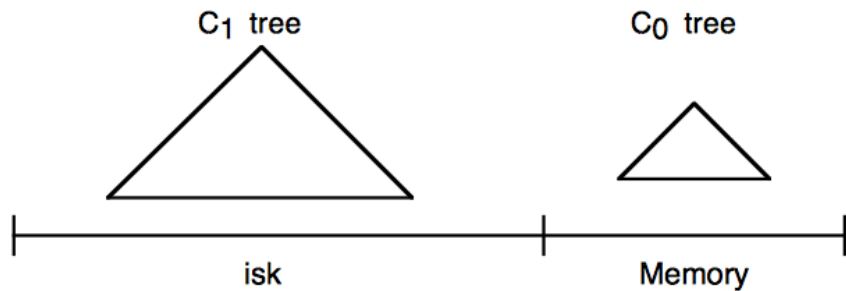


Figure 2.1. Schematic picture of an LSM-tree of two components

(Patrick O'Neil, The Log-Structured Merge Tree, 1996)

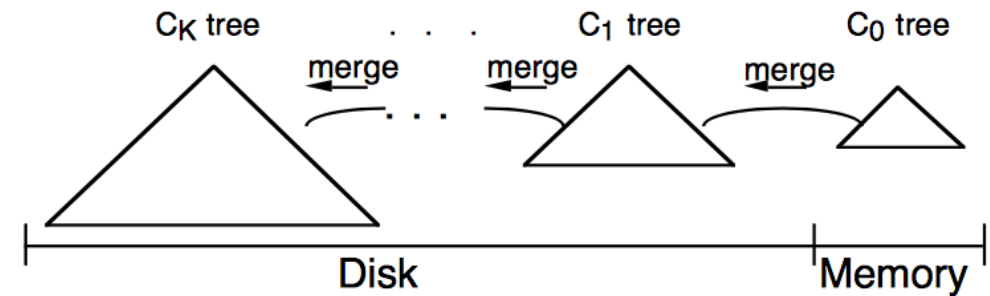
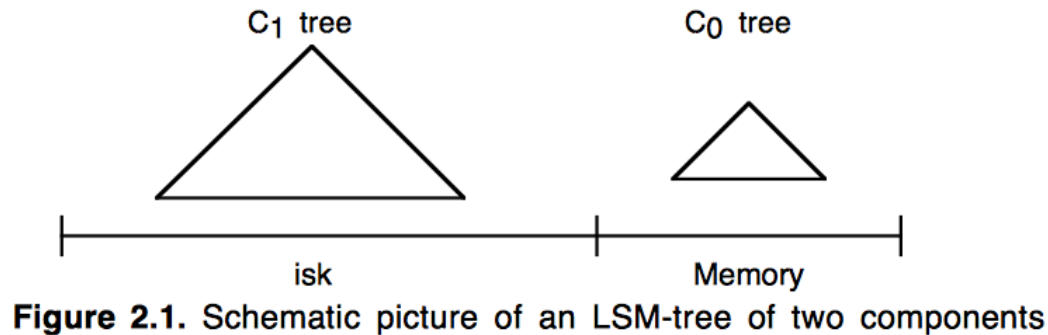


Figure 3.1. An LSM-tree of $K+1$ components

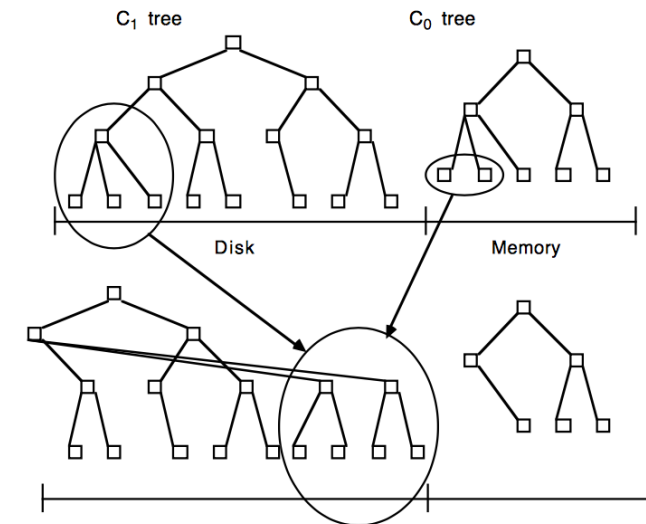
(Patrick O'Neil, The Log-Structured Merge Tree, 1996)

LSM-tree

- What is LSM-tree
 - Rolling Merge
 - Merge sort levels onto subsequent levels for deleting old data
 - All data in sorted order



(Patrick O'Neil, The Log-Structured Merge Tree, 1996)



(Patrick O'Neil, The Log-Structured Merge Tree, 1996)

Architecture

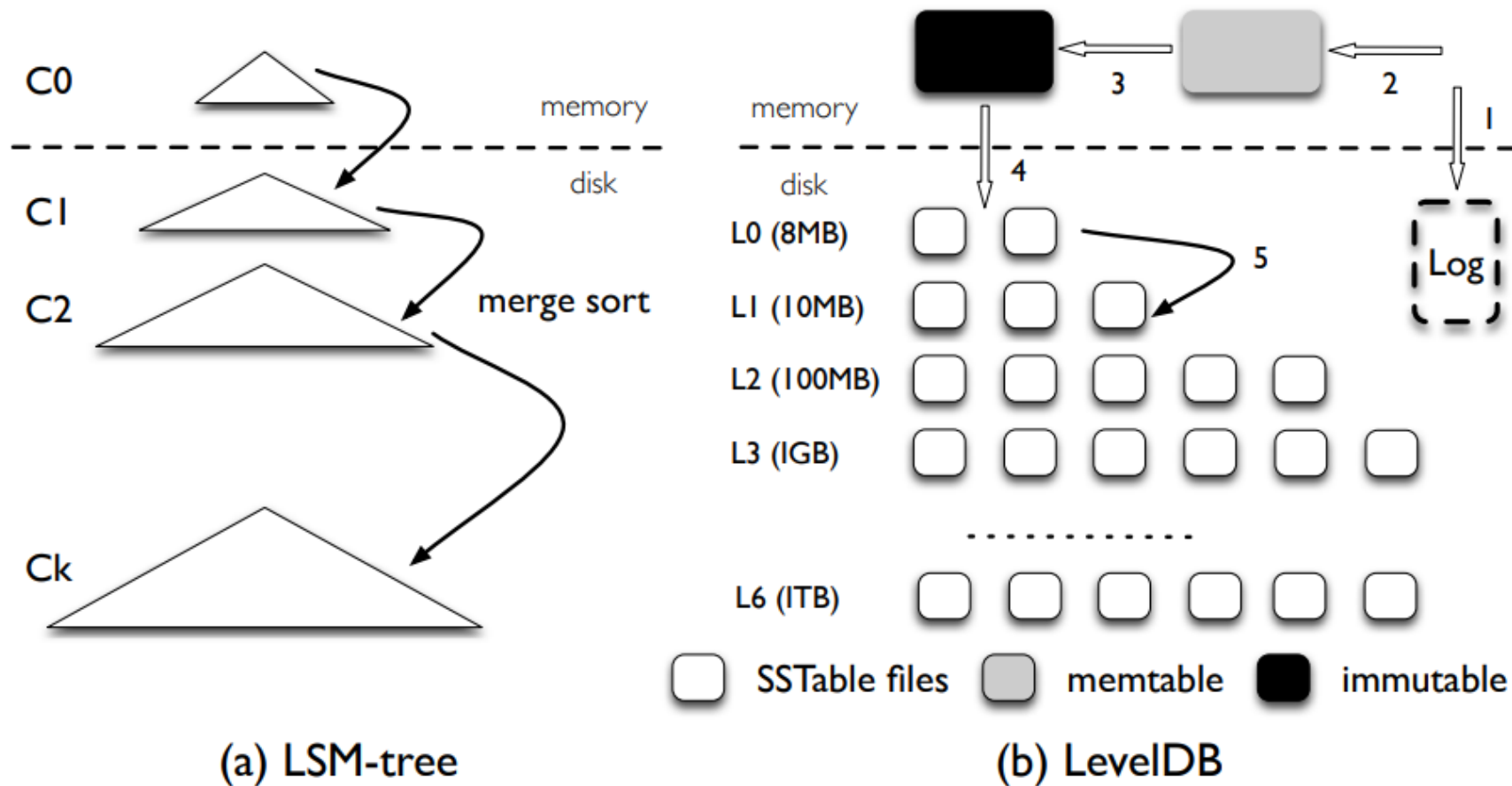


Figure 1: **LSM-tree and LevelDB Architecture.** *This*

Lanyue Lu, WiscKey (Fast '16)

Architecture

- Real implementation in LevelDB
 - Memtable for C0
 - Further separated into mutable and immutable
 - Managed by the Skiplist data structure (or hash)
 - A set of SSTables for C1~Ck (multiple Levels, configurable)
 - Default fanout ratio = 10, $|L_{i+1}| / |L_i|$
 - SSTable internals
 - ✓ data block, index block (logically B+-tree)
- Log (WAL) for durability
 - A set of records where each record consists of CRC, size, type and payload

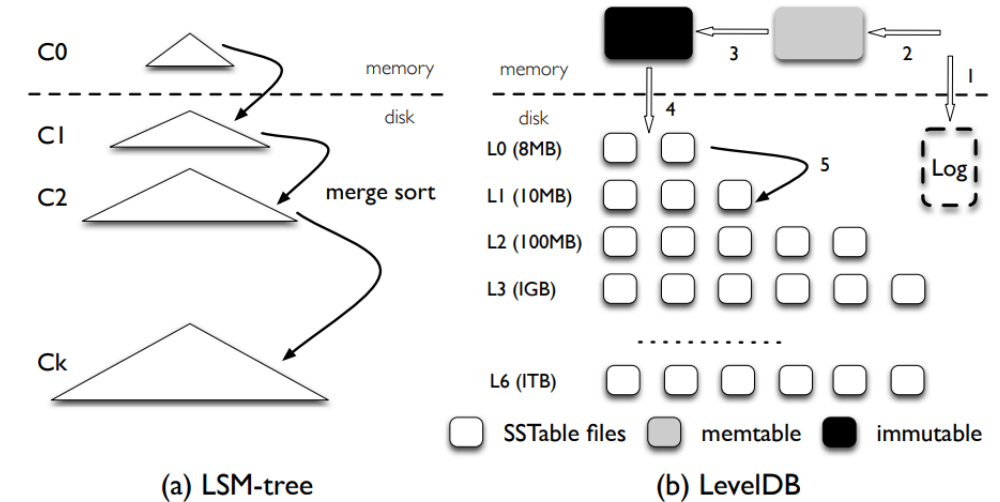


Figure 1: **LSM-tree and LevelDB Architecture.**

Lanyue Lu, WiscKey (Fast '16)

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- Key-Value Interface / Operations
 - ✓ Key-Value Interface
 - Put/Delete, Get, Seek(iterator)
 - ✓ Internal Operations
 - Flush, Compaction

Key-Value Interface

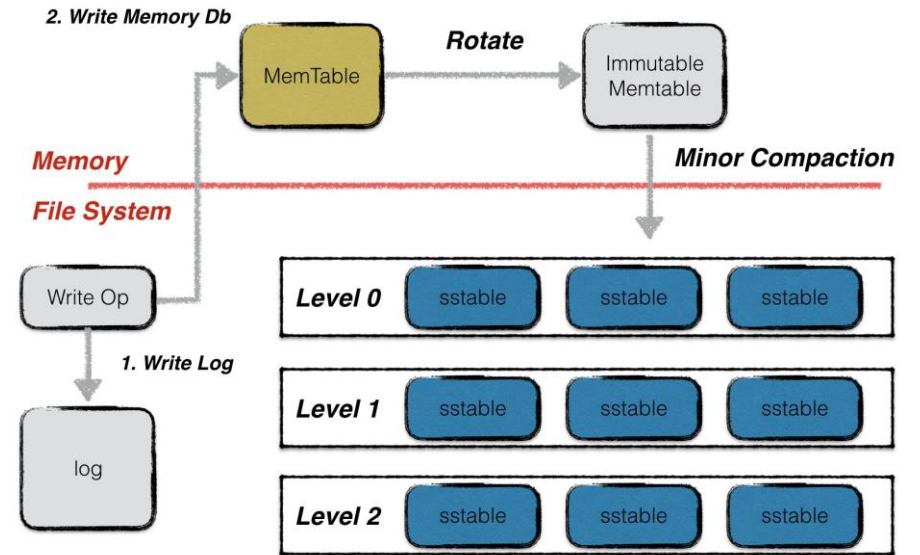
- Key/Value: Arbitrary byte streams
- User visible interfaces
 - put, get, range scan(iterator), delete, single delete, ...

```
std::string value;  
leveldb::Status s = db->Get(leveldb::ReadOptions(), key1, &value);  
if (s.ok()) s = db->Put(leveldb::WriteOptions(), key2, value);  
if (s.ok()) s = db->Delete(leveldb::WriteOptions(), key1);
```

```
leveldb::Iterator* it = db->NewIterator(leveldb::ReadOptions());  
for (it->SeekToFirst(); it->Valid(); it->Next()) {  
    cout << it->key().ToString() << ": " << it->value().ToString() << endl;  
}  
assert(it->status().ok()); // Check for any errors found during the scan  
delete it;
```

Put/Delete

- Put
 - Insert into memtable
 - Batch write
- Delete
 - Insert a delete marker to the memtable
 - annihilates with the value during compaction.



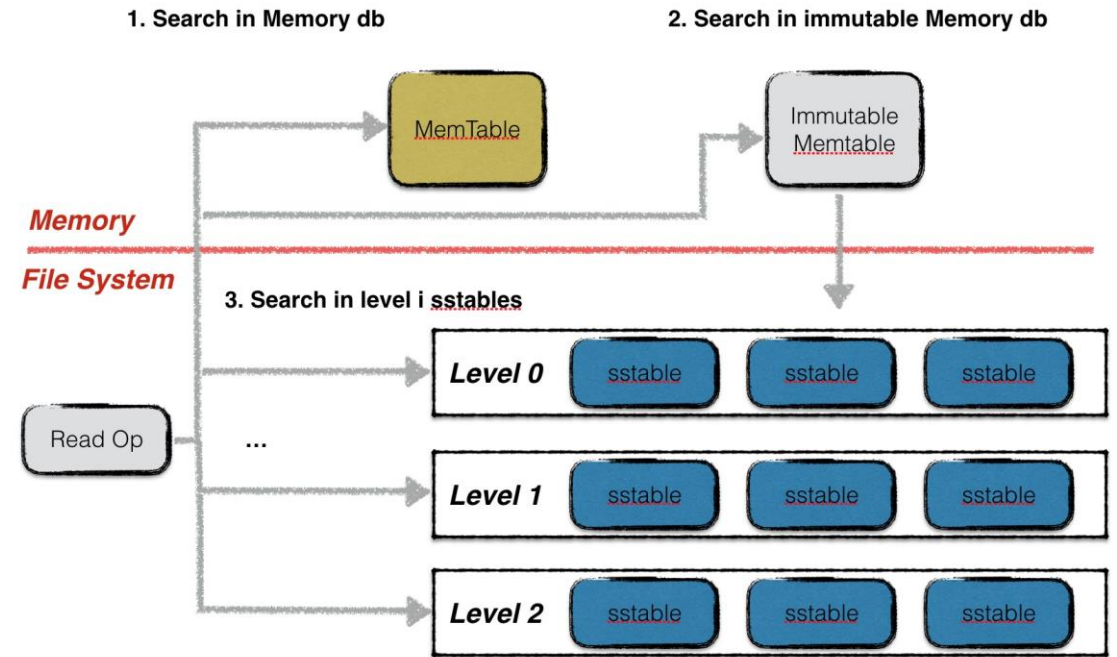
<https://leveldb-handbook.readthedocs.io/zh/latest/>

SequenseNumber (uint64)	count (uint32)	record0	recordN
----------------------------	-------------------	---------	-------	---------

ValueType (char)	key_len (varint32)	key_data (key_len)	value_len (varint32)	value_data (value_len)
---------------------	-----------------------	-----------------------	-------------------------	---------------------------

Get

1. Memtable
2. Immutable Memtable
3. SSTables
 - Bloom filter
 - Cache



<https://leveldb-handbook.readthedocs.io/zh/latest/>

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- Key-Value Interface / Operations
 - ✓ Key-Value Interface
 - Put/Delete, Get, Seek(iterator)
 - ✓ Internal Operations
 - Flush, Compaction

Seek

1. (Merge) Iterator

- Memtable
- Immutable Memtable
- SSTable in each levels

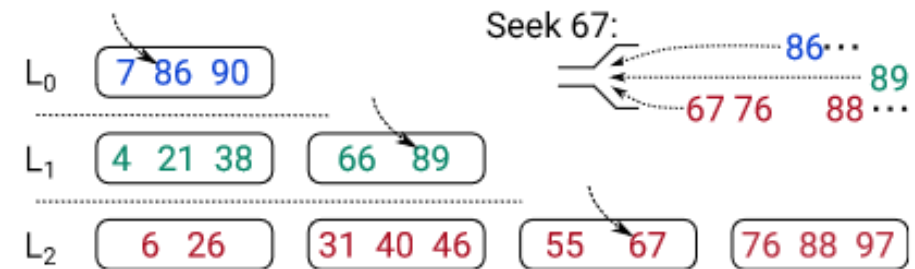
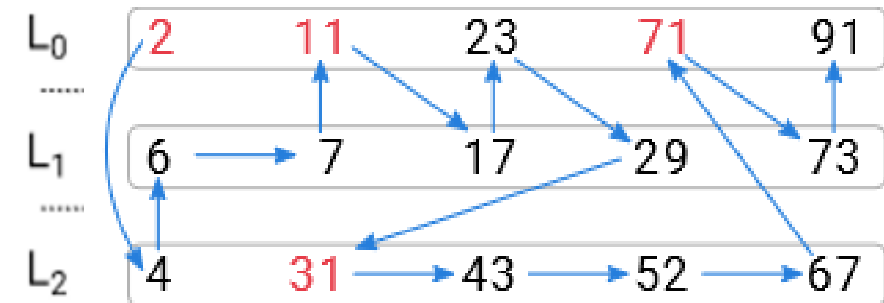


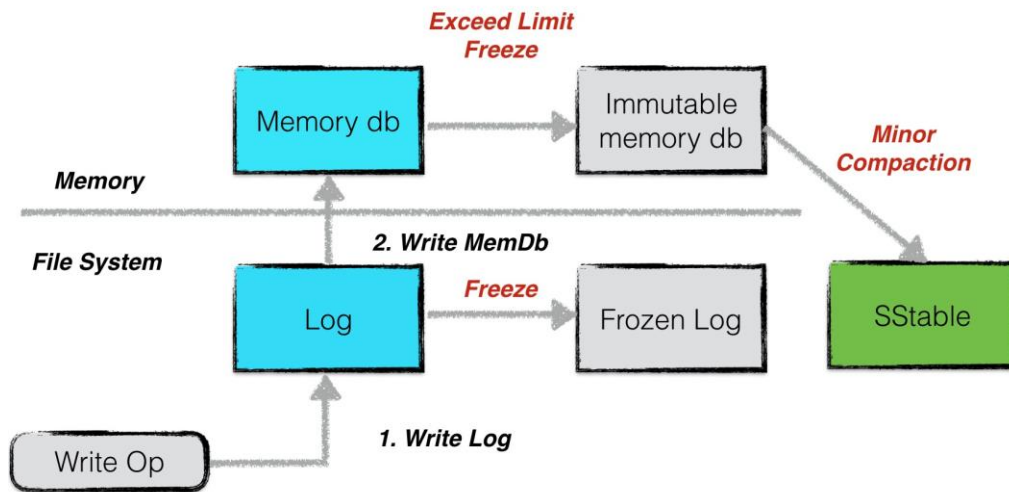
Figure 1: An LSM-tree using leveled compaction

2. Iterate until done



WAL

- Write Ahead Log

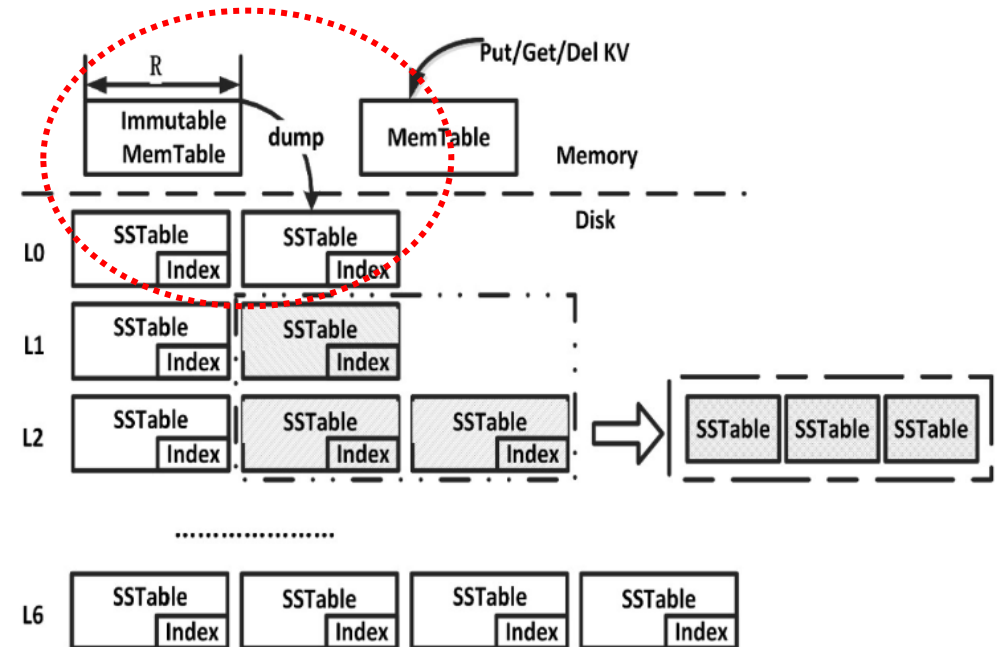


<https://leveldb-handbook.readthedocs.io/zh/latest/>

```
-rw-r--r-- 1 root root 37922501 7월 1 15:44 000214.sst
-rw-r--r-- 1 root root 37920200 7월 1 15:44 000216.sst
-rw-r--r-- 1 root root 37910828 7월 1 15:44 000219.sst
-rw-r--r-- 1 root root 37906740 7월 1 15:44 000221.sst
-rw-r--r-- 1 root root 37905482 7월 1 15:44 000224.sst
-rw-r--r-- 1 root root 37909294 7월 1 15:44 000227.sst
-rw-r--r-- 1 root root 66217640 7월 1 15:44 000228.log
-rw-r--r-- 1 root root 37892964 7월 1 15:44 000229.sst
-rw-r--r-- 1 root root 18621323 7월 1 15:44 000231.log
-rw-r--r-- 1 root root      16 7월 1 15:43 CURRENT
-rw-r--r-- 1 root root      37 7월 1 15:43 IDENTITY
-rw-r--r-- 1 root root       0 7월 1 15:43 LOCK
-rw-r--r-- 1 root root 572956 7월 1 15:44 LOG
-rw-r--r-- 1 root root 14592 7월 1 15:44 MANIFEST-000004
-rw-r--r-- 1 root root 6180 7월 1 15:43 OPTIONS-000007
```

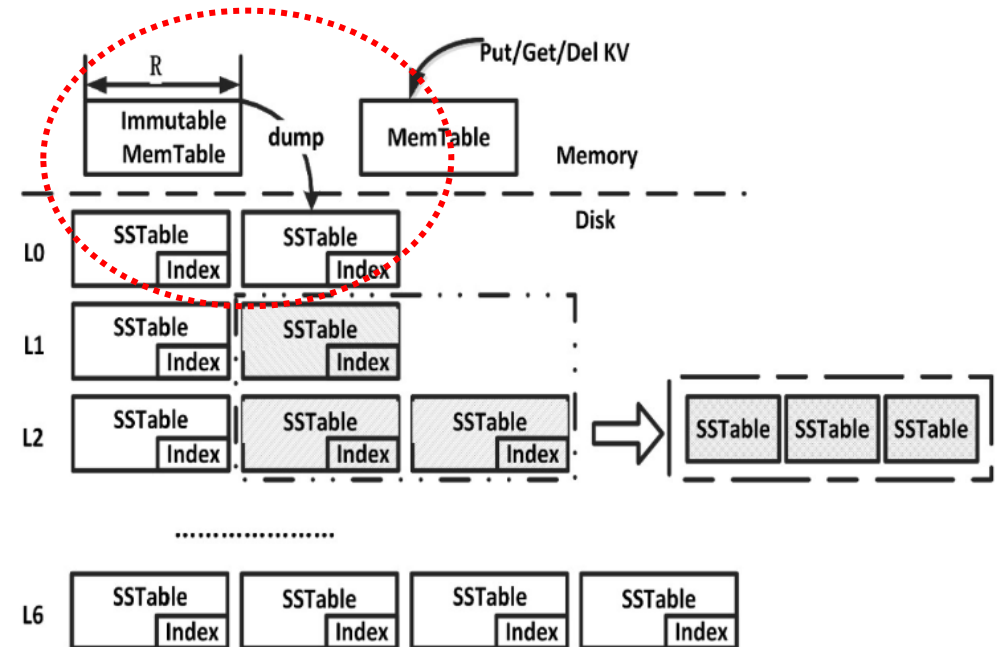
Flush

- Flush
 - Dump immutable into SST file
 - Trivial Move
 - Push as further down as possible if
 - ✓ No overlap with current level
 - ✓ Overlapping with no more than 10 SSTs in the next level



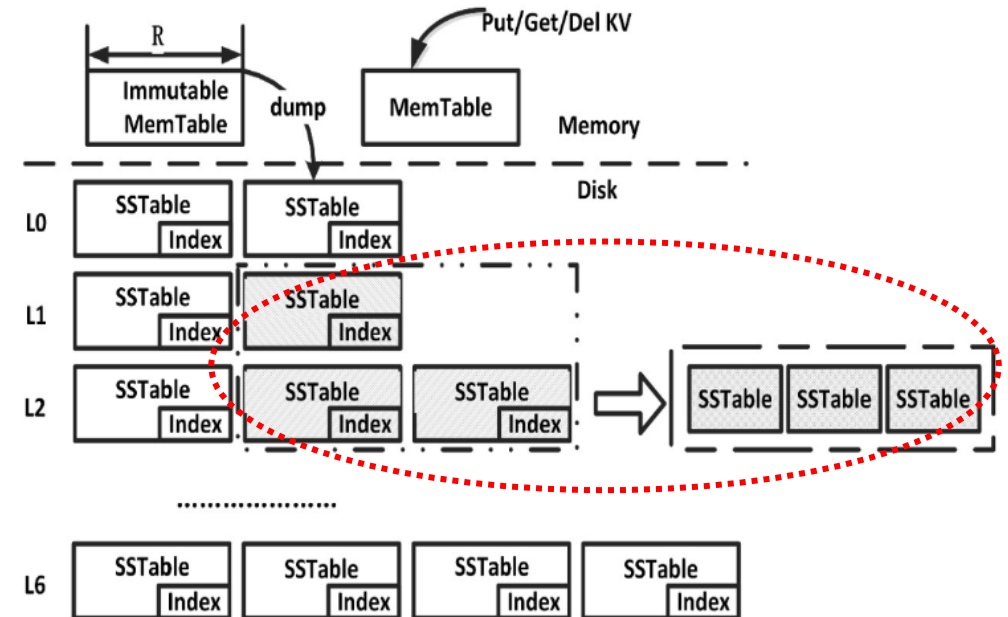
Flush

- Trigger (both must be satisfied)
 - memtable exceeds 4MB
 - immutable = NULL
- Procedure
 - Block write thread
 - move memtable to immutable
 - create new memtable.
 - Background flushing for immutable



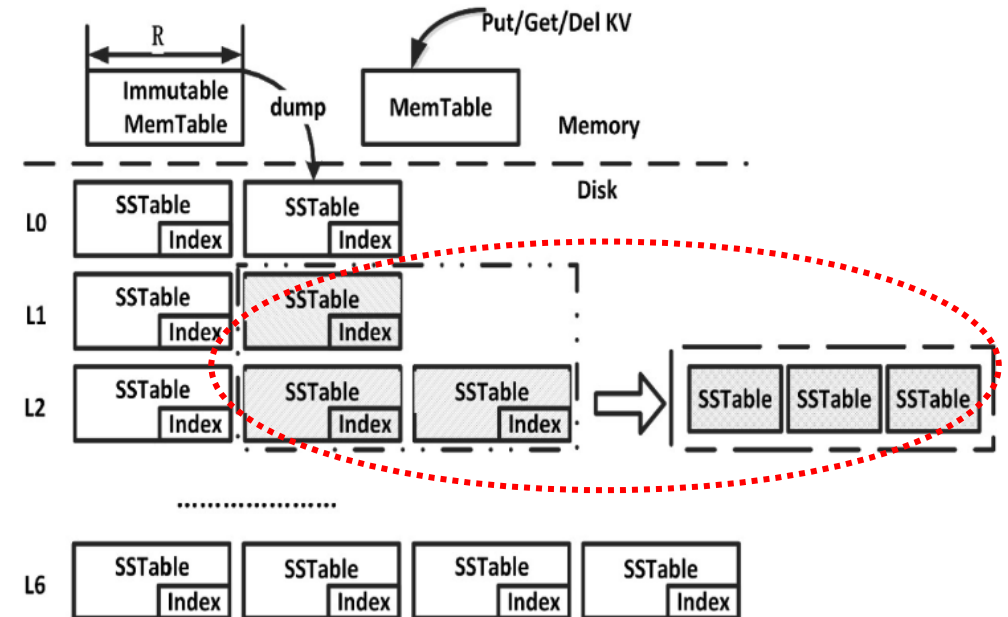
Compaction

- Trigger (at least one of the conditions listed is satisfied)
 - L0 SST exceeds 8
 - L_i ($i > 0$) SST space exceeds 10^i MB
 - allowed_seek used up when calling Get() (RAF)
 - Manual Compaction
- Check During
 - DB open, Write, Get, Compaction



Compaction

- Procedure
 - MaybeScheduleCompaction
 - BackgroundWork
 - PickCompaction
 - Pick level and SST
 - Pick same level's overlapped SSTs
 - Pick next level's overlapped SSTs
 - DoCompaction
 - Merge Iterator
 - ✓ merge sort, garbage collection
 - Write new SSTs (max file size)



1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- Data Structure

- ✓ Slice, Key
- ✓ Log
- ✓ Memtable
- ✓ SSTable
- ✓ Bloom Filter
- ✓ Cache
- ✓ Manifest

Slice

- `leveldb::Slice`
 - return value of `it->key()`, `it->value()`
- a simple structure that contains a length and a pointer to an external byte array.
- Slice is cheaper than `std::string`
 - not need to copy potentially large keys and values.
- Does not return null-terminated C-style strings
 - leveldb keys and values are allowed to contain `'\0'` bytes.

```
class Slice {  
    ...  
    private:  
    const char* data_;  
    size_t size_;  
};
```

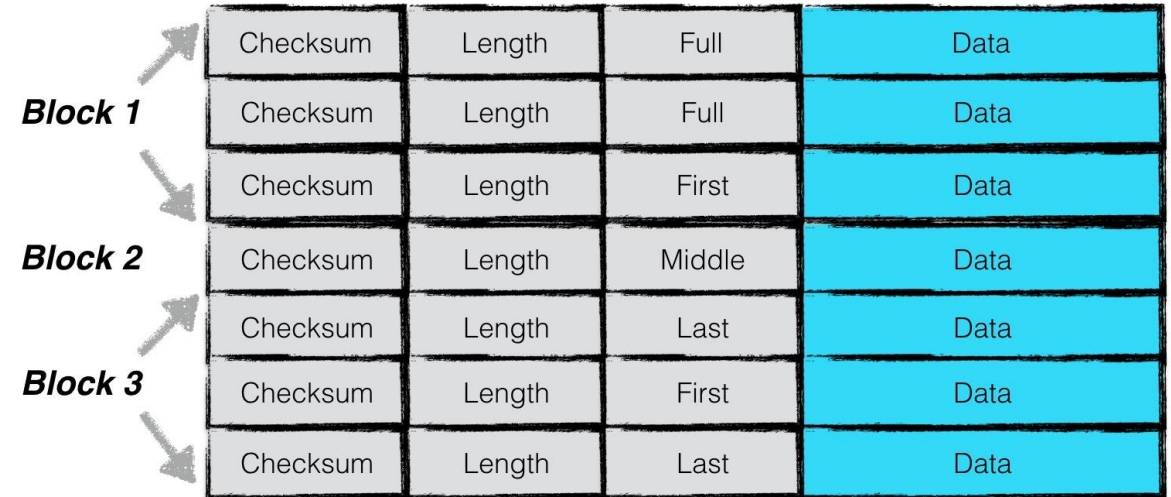
Key

- Value Type
 - Value, Deletion
- Sequence Number
 - every put/delete operation has a sequence number
 - 64bits global variable
 - compaction, snapshot depends on the sequence number.
- Key
 - userkey: passed from user, in Slice format
 - InternalParsedKey: userkey + seqNum + valuetype
 - InternalKey: string representation of InternalParsedKey

```
enum ValueType {  
    kTypeDeletion = 0x0,  
    kTypeValue = 0x1  
};
```

Log

- File
 - *.log: WAL file
 - LOG: not a WAL file, text info log
- Block (32KB)
 - Sequence of chunks
- Chunk header
 - checksum (4B), len(2B), type(1B),
 - Types of chunk
 - Full / First / Middle / Last

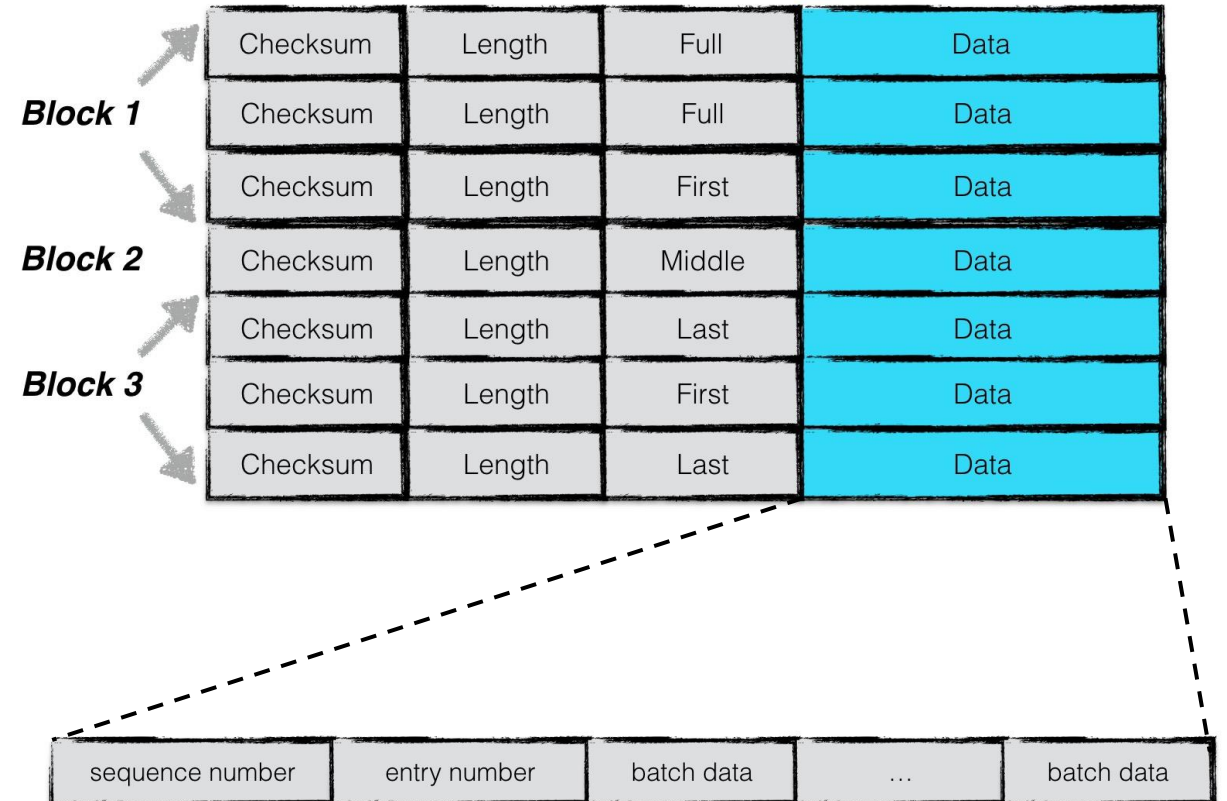


<https://leveldb-handbook.readthedocs.io/zh/latest/>

```
mingu@sever: /tmp/leveldbtest-1000/dbbench$ cat LOG
2022/07/06-15:01:31.954351 139755705333568 Creating DB /tmp/leveldbtest-1000/
2022/07/06-15:01:31.969198 139755705333568 Delete type=3 #1
2022/07/06-15:01:32.016174 139755618563840 Level-0 table #5: started
2022/07/06-15:01:32.029223 139755618563840 Level-0 table #5: 1887662 bytes OK
2022/07/06-15:01:32.032375 139755618563840 Delete type=0 #3
```

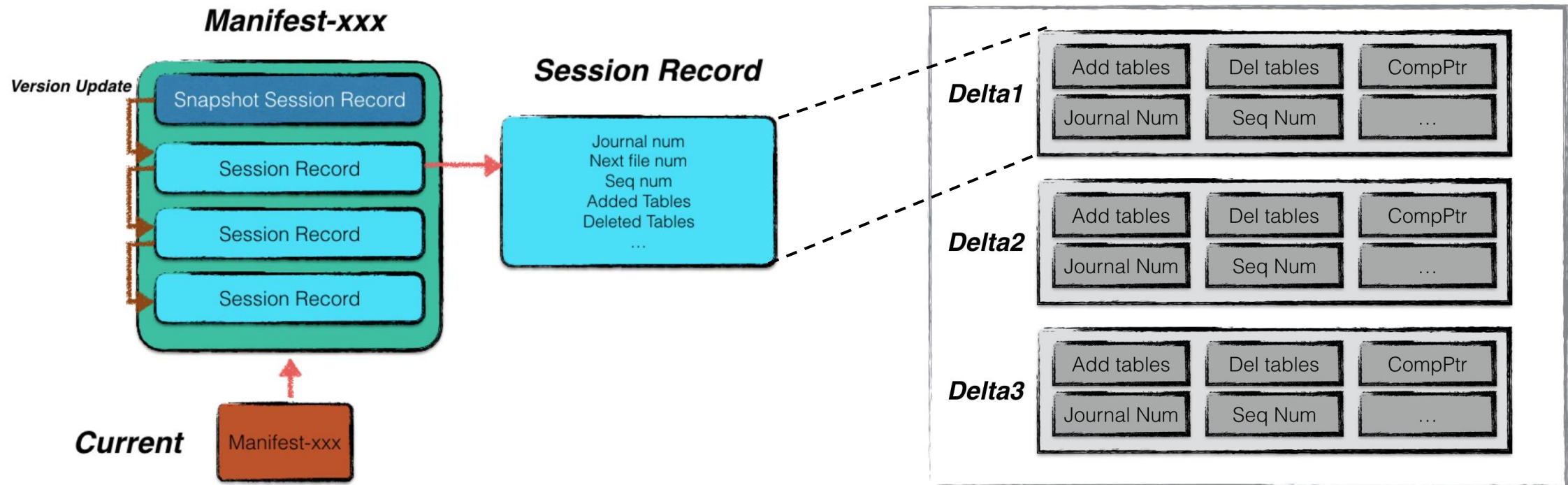
Log

- Log Data (Contents)
 - Header
 - Sequence number
 - Num of entries
 - Data
 - Batch data



<https://leveldb-handbook.readthedocs.io/zh/latest/>

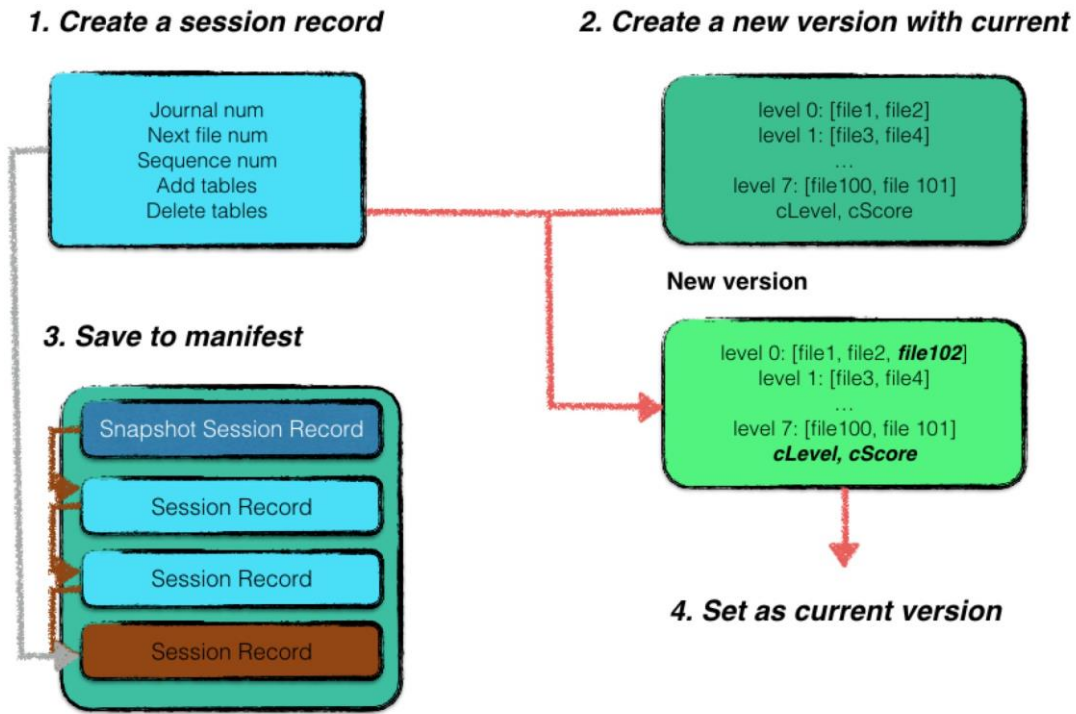
Manifest



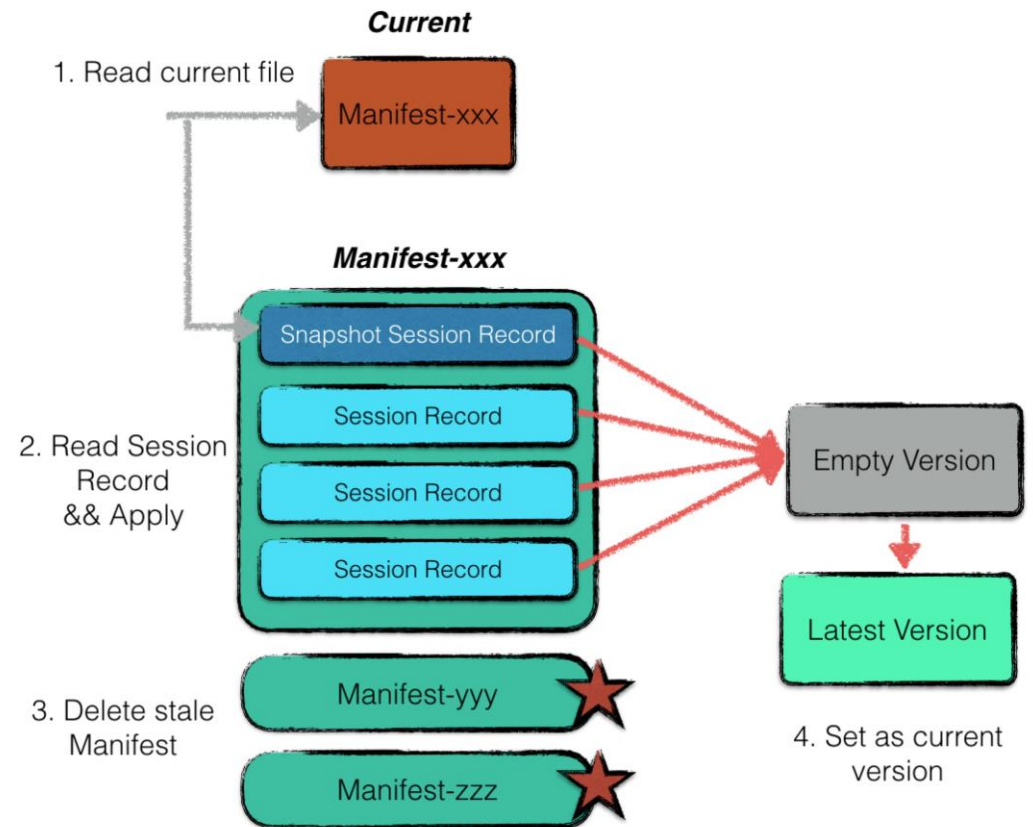
<https://leveldb-handbook.readthedocs.io/zh/latest/>

Manifest

Commit



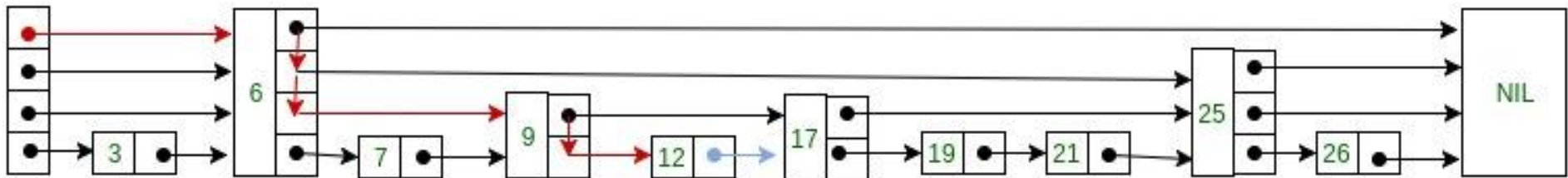
Recovery (DB open)



<https://leveldb-handbook.readthedocs.io/zh/latest/>

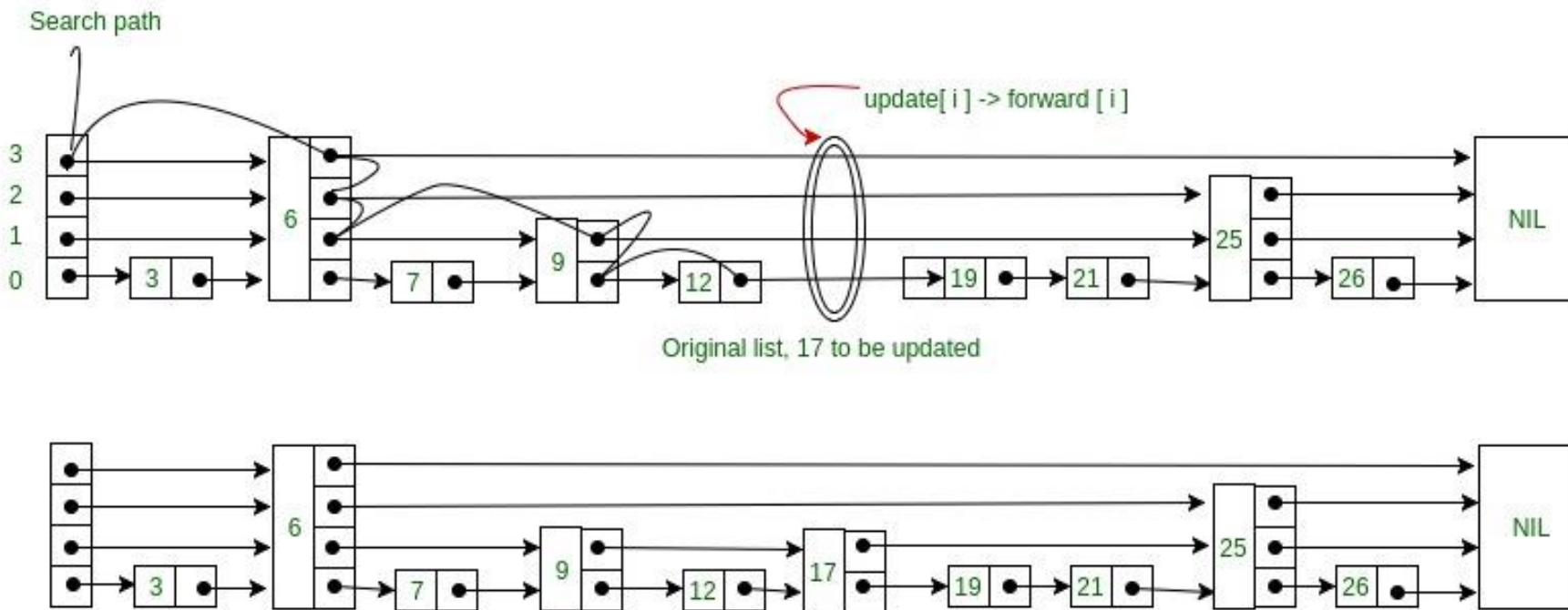
Memtable

- KV pairs in memory, managed by Skiplist
- Skiplist: a data structure with a set of sorted linked lists
 - All keys appears in the last list
 - Some keys also appear in the upper list (for fast search)
- Good for both lookup and scan
 - Get the benefits of both Binary Tree and List
 - Useful in multi - threaded system architectures



Memtable

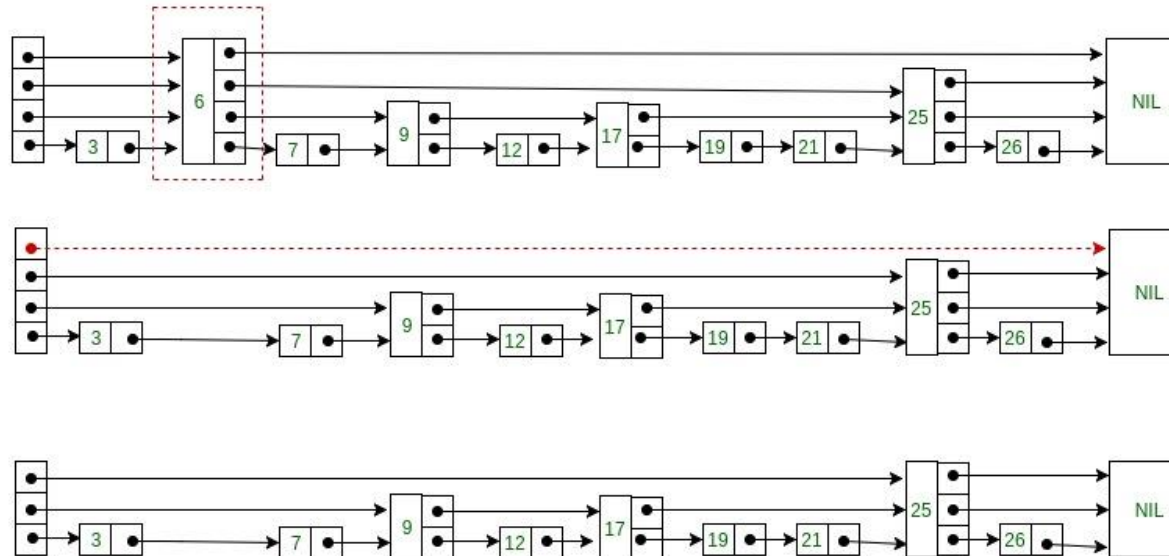
- Skiplist
 - (Search 17) -> Insert 17



<https://www.geeksforgeeks.org/skip-list-set-2-insertion/?ref=lbp>

Memtable

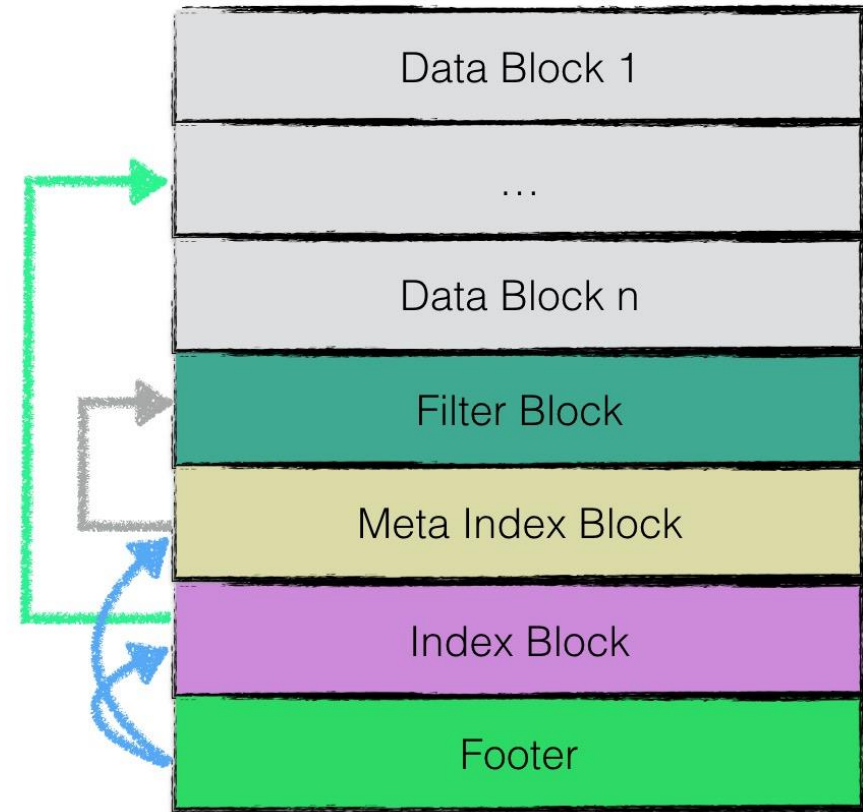
- Skiplist
 - Delete 6
 - No deletion in LevelDB
 - Just out-place update with sequence number



<https://www.geeksforgeeks.org/skip-list-set-3-searching-deletion/?ref=lbp>

SSTable

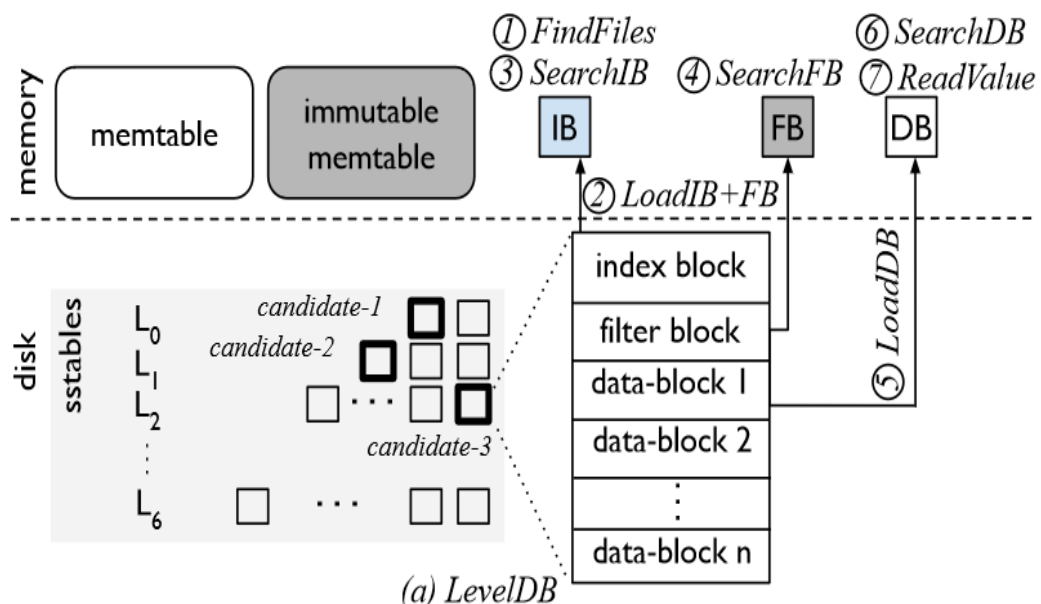
- Structure
 - Data block
 - Filter block
 - Bloom filter
 - Meta index block
 - Index of filter block
 - Index block
 - Index of data block
 - Footer
 - Index of meta-index / index



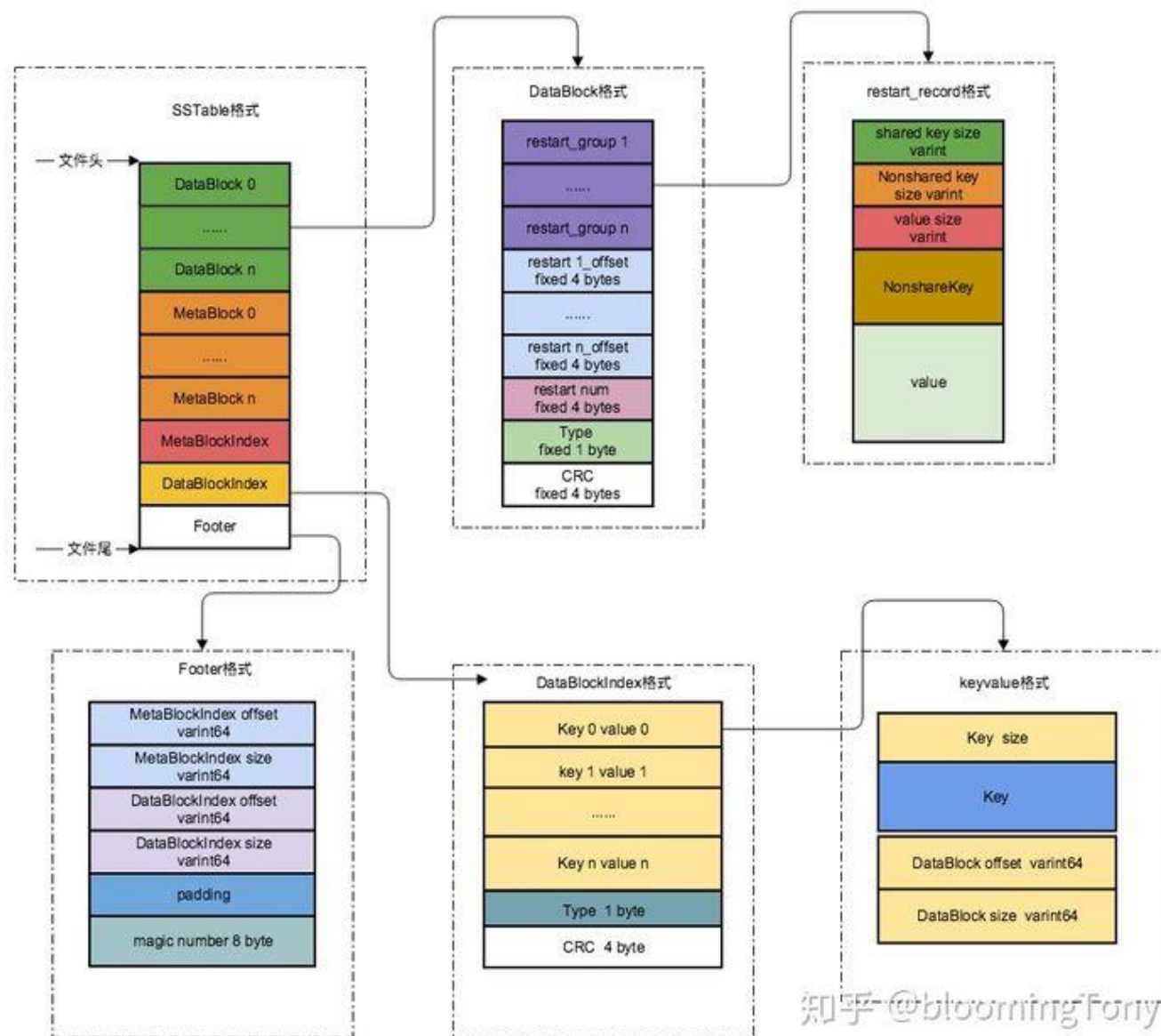
<https://leveldb-handbook.readthedocs.io/zh/latest/>

SSTable

Read KV from SST



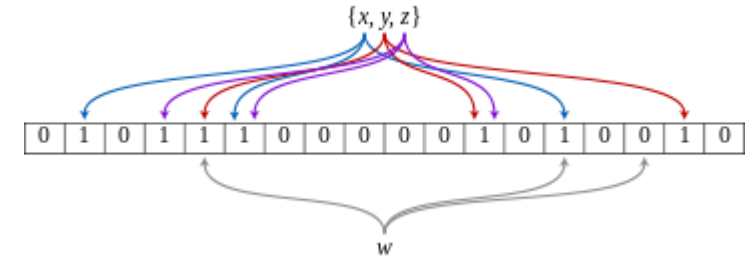
Yifan Dai, From WiscKey to Bourbon, OSDI '20



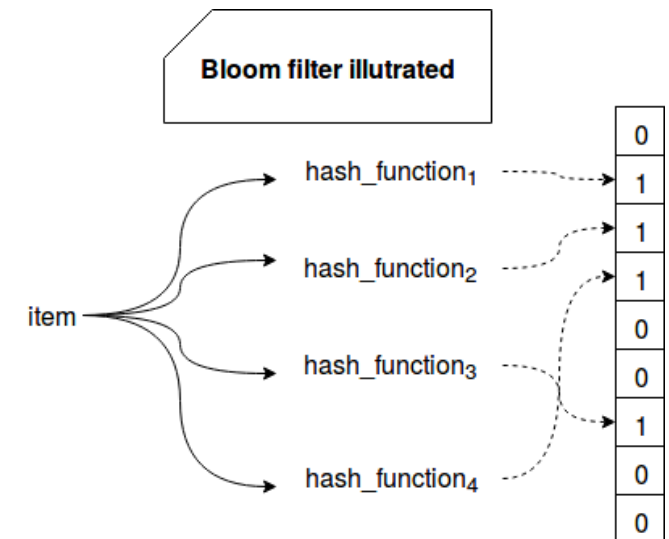
<https://zhuanlan.zhihu.com/p/37633790>

Bloom Filter

- Used to reduce the read amplification (unnecessary read)
- Bloom filter: a data structure for identifying membership
 - Based on bits and multiple hashes
 - Good property: No false negative
 - Issue: can yield false positives
 - tradeoffs between bits and rate
 - 1% false positive rate with 9.9 bits per key, from RocksDB wiki
- Not only SSTable (per SSTable or per block) but also Memtable

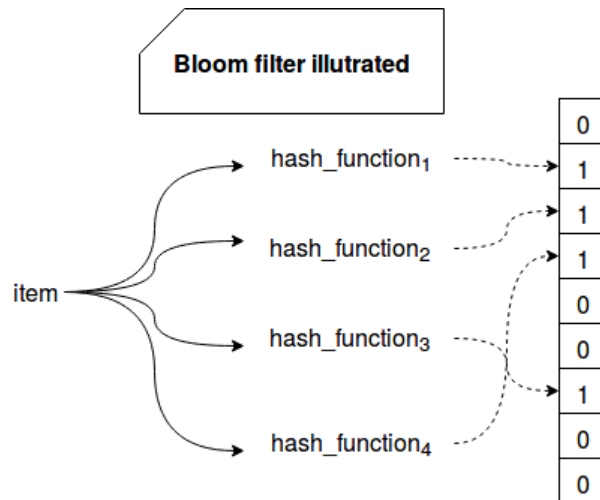


https://en.m.wikipedia.org/wiki/File:Bloom_filter.svg



<https://www.waitingforcode.com/big-data-algorithms/bloom-filter/read>

Bloom Filter



<https://www.waitingforcode.com/big-data-algorithms/bloom-filter/read>

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

<https://commons.wikimedia.org/wiki/File:ConfusionMatrixRedBlue.png>

1. Add items to bloom filter using a hash function



2. Bloom filter containing "Monday," "Thursday," "Friday"



3. To look up "Saturday," generate a hash for "Saturday" ...



4. ...and compare to the bloom filter



Yes

Yes

No

"Saturday" is not in the set of items in the bloom filter.

<https://numenta.com/blog/2012/10/07/wait-the-brain-is-a-bloom-filter/>

Bloom Filter

Bloom Filters by Example

A Bloom filter is a data structure designed to tell you, rapidly and memory-efficiently, whether an element is present in a set.

The price paid for this efficiency is that a Bloom filter is a **probabilistic data structure**: it tells us that the element either *definitely is not* in the set or *may be* in the set.

The base data structure of a Bloom filter is a **Bit Vector**. Here's a small one we'll use to demonstrate:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

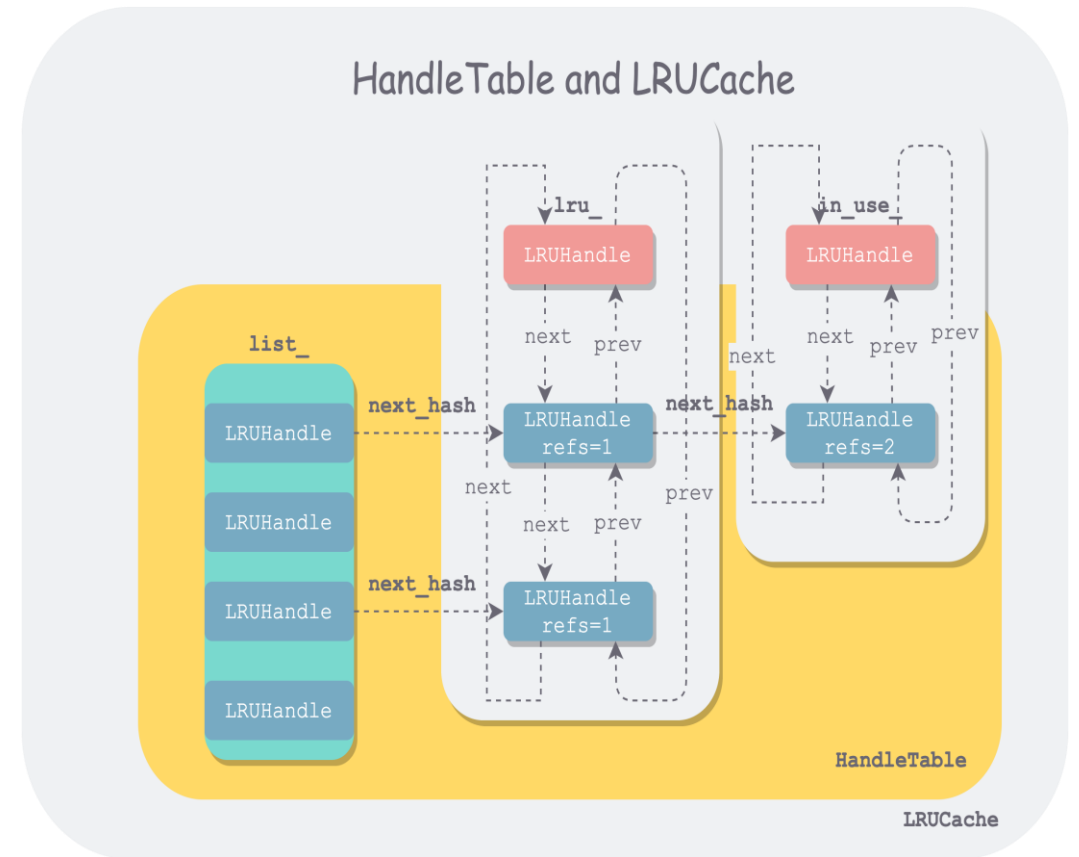
Each empty cell in that table represents a bit, and the number below it its index. To add an element to the Bloom filter, we simply hash it a few times and set the bits in the

Online bloom filter practice, excellent visualization

<https://lilimlib.github.io/bloomfilter-tutorial/>

Cache

- Cache uncompressed block
- Multi-threading support
- Data structure
 - Hash + array + 2 double linked list
 - 2 Double linked list
 - In_use_, lru_

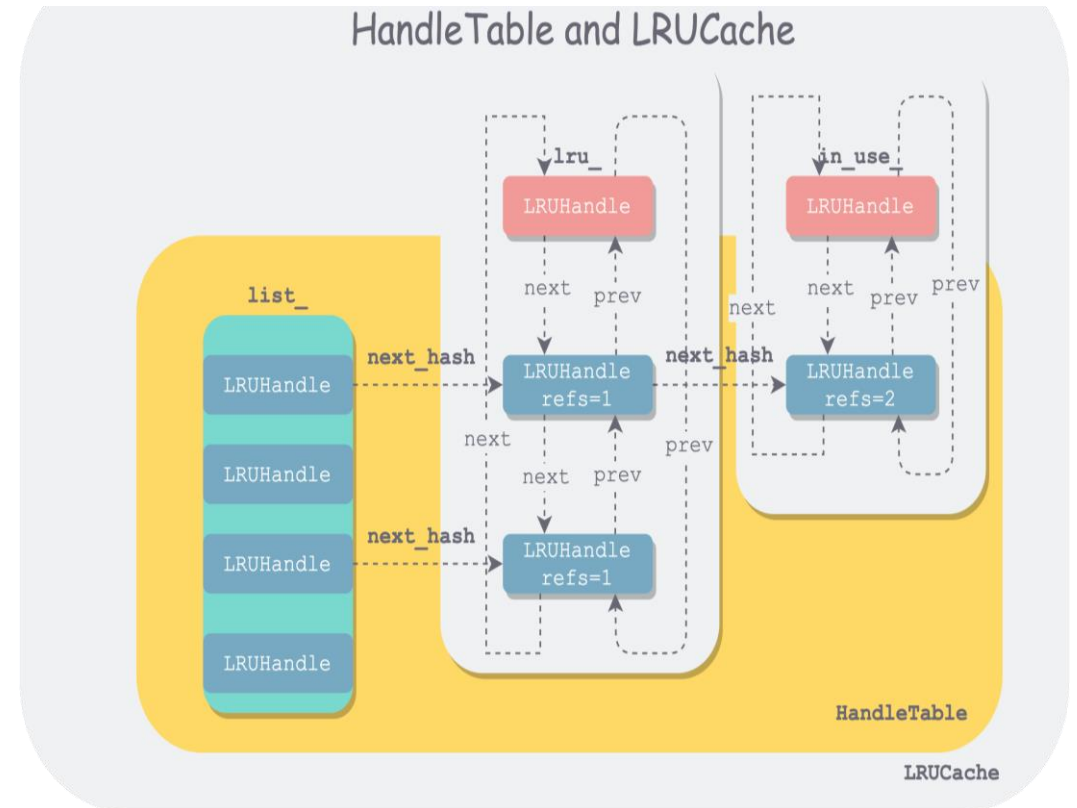
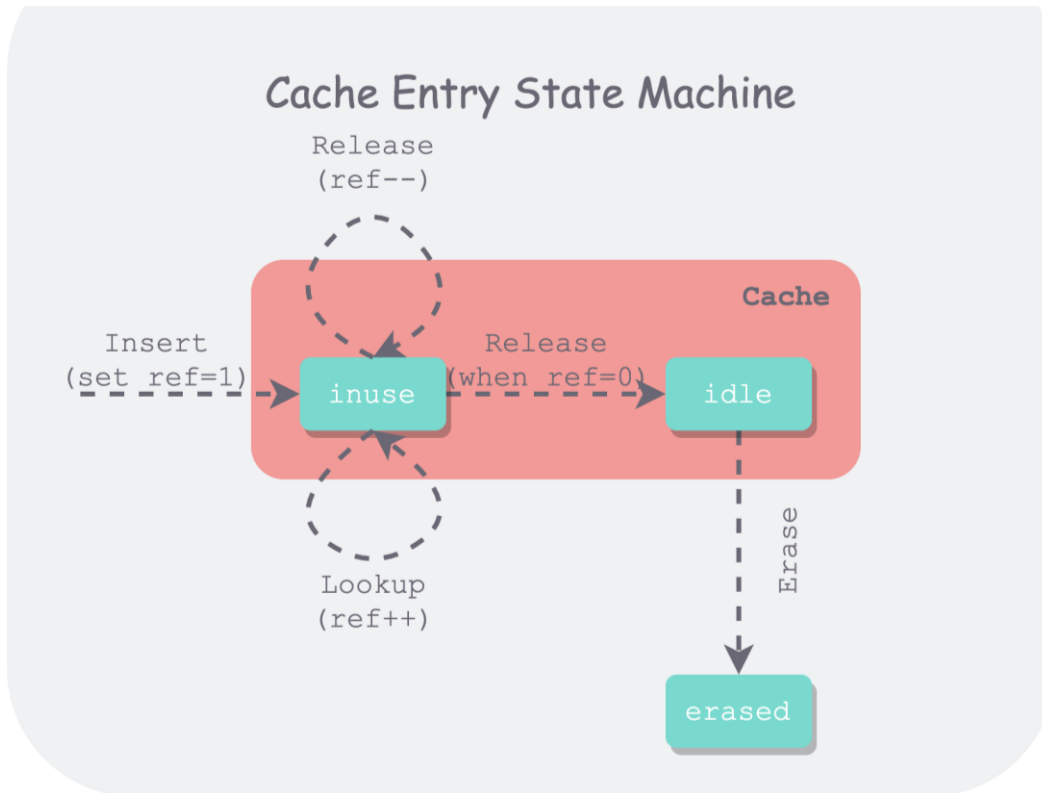


[木鸟杂记, Talking about LevelDB data structure \(CHS\), 2021](#)

Cache

■ Ref

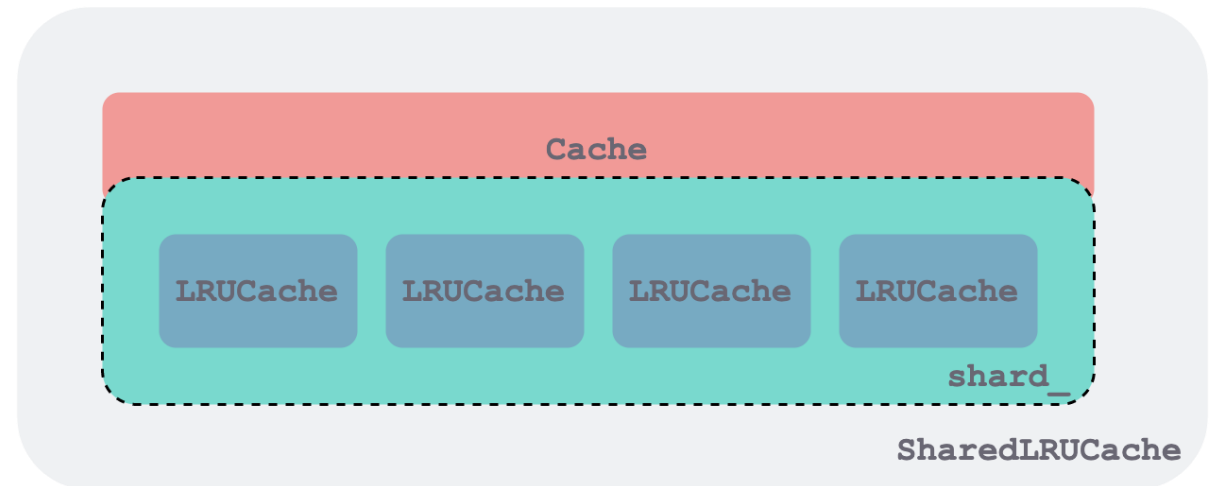
- $\text{ref} = 2 \text{ (in_use_ \&\& lru_)} / \text{ref} = 1 \text{ (!in_use_ \&\& lru_)} / \text{ref} = 0 \text{ (!in_use_ \&\& !lru_)}$



[木鸟杂记, Talking about LevelDB data structure \(CHS\), 2021](#)

Cache

- Lock
 - Mutex_
- Sharding
 - For locking granularity
 - coarse/fine grained
 - 16 shards



[木鸟杂记, Talking about LevelDB data structure \(CHS\), 2021](#)

- 1. LevelDB Architecture
- 2. Key-Value Interface
- 3. Internal Operations
- 4. Data Structure
- 5. LevelDB Installation
- 6. db_bench experiment
- 7. References
- 8. Homework

■ LevelDB Installation

✓ LevelDB install

- Release Mode
- Debug Mode

✓ db_bench

- Benchmark, Option, Command
- Shell script

LevelDB Install

- Install LevelDB twice separately
 - Release mode
 - Experiment, Benchmark
 - Debug mode
 - Analysis

Release Mode

- Installation (posix)
 - Install guide is available at LevelDB repository
 - <https://github.com/google/leveldb>

```
$ sudo apt-get update
```

```
$ sudo apt-get install build-essential
```

```
$ sudo apt-get install cmake
```

```
$ git clone --recurse-submodules https://github.com/google/leveldb.git leveldb_
```

```
$ cd leveldb_release
```

```
$ mkdir -p build && cd build
```

```
$ cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build .
```

- Test
 - ./db_bench (dir: leveldb_release/build)

Getting the Source

```
git clone --recurse-submodules https://github.com/google/leveldb.git
```

Building

This project supports CMake out of the box.

Build for POSIX

Quick start:

```
mkdir -p build && cd build  
cmake -DCMAKE_BUILD_TYPE=Release .. && cmake --build .
```

<https://github.com/google/leveldb/blob/main/README.md>

Debug Mode

- Installation (posix)

```
$ git clone --recurse-submodules \
  https://github.com/google/leveldb.git leveldb_debug
$ cd leveldb_debug
```

- Add '-g', '-pg' gcc option on leveldb/CMakeList.txt

- -g : debug option
- -pg : gprof option
- set(CMAKE_CXX_FLAGS "\${CMAKE_CXX_FLAGS} -g -pg")

- Build

```
$ mkdir -p build && cd build
$ cmake -DCMAKE_BUILD_TYPE=Debug .. && cmake --build .
```

- Test

```
$ ./db_bench (dir: leveldb_debug/build)
```

2. Add -g -pg option on leveldb/CMakeList.txt

```
+ # Add -g -pg option for gdb, ufttrace
+ set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -g -pg")

if(CMAKE_CXX_COMPILER_ID STREQUAL "MSVC")
  # Disable C++ exceptions.
```

-g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

-pg Generate extra code to write profile information suitable for the analysis program **prof** (for **-p**) or **gprof** (for **-pg**). You must use this option when compiling the source files you want data about, and you must also use it when linking.

gcc(1) — Linux manual pag
(<https://man7.org/linux/man-pages/man1/gcc.1.html>)

db_bench

- LevelDB internal benchmark tool
 - Metrics
 - Throughput, latency, size ...
 - Trace code
 - gdb, uftace

```
mingu@sever:~/leveldb_release/build$ ./db_bench
LevelDB:    version 1.23
Date:       Fri Jul  8 21:38:00 2022
CPU:        16 * Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
CPUCache:   16384 KB
Keys:        16 bytes each
Values:      100 bytes each (50 bytes after compression)
Entries:     1000000
RawSize:     110.6 MB (estimated)
FileSize:    62.9 MB (estimated)

-----
fillseq      :      1.159 micros/op;   95.5 MB/s
fillsync     :    2441.636 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :      2.259 micros/op;   49.0 MB/s
overwrite    :      2.833 micros/op;   39.1 MB/s
readrandom   :      2.795 micros/op; (864322 of 1000000 found)
readrandom   :      2.336 micros/op; (864083 of 1000000 found)
readseq      :      0.108 micros/op; 1020.7 MB/s
readreverse  :      0.166 micros/op;  666.8 MB/s
compact      :   528715.000 micros/op;
readrandom   :      1.821 micros/op; (864105 of 1000000 found)
readseq      :      0.093 micros/op; 1183.2 MB/s
readreverse  :      0.149 micros/op;  740.8 MB/s
fill100K     :      747.593 micros/op;  127.6 MB/s (1000 ops)
crc32c       :      0.868 micros/op; 4502.8 MB/s (4K per op)
snappycomp   :      2.273 micros/op; 1718.3 MB/s (output: 55.1%)
snappyuncomp :      0.385 micros/op; 10147.6 MB/s
```

Environment setup

(0) root mode

\$ sudo su

(1) disable swap entirely to avoid performance problems and inconsistencies

\$ swapoff --all

(2) disable zone_reclaim_mode

\$ echo 0 > /proc/sys/vm/zone_reclaim_mode

(3) maximum open file

\$ sysctl fs.file-max

\$ sysctl -w fs.file-max=5000000

(4) Log out

\$ exit

```
mingu@sever:~/leveldb_release/build$ sudo su
root@sever:/home/mingu/leveldb_release/build# swapoff --all
root@sever:/home/mingu/leveldb_release/build# echo 0 > /proc/sys/vm/zone_reclaim_mode
root@sever:/home/mingu/leveldb_release/build# sysctl fs.file-max
fs.file-max = 9223372036854775807
root@sever:/home/mingu/leveldb_release/build# sysctl -w fs.file-max=5000000
fs.file-max = 5000000
root@sever:/home/mingu/leveldb_release/build# sysctl -w fs.file-max=9223372036854775807
fs.file-max = 9223372036854775807
```

Options

- All db_bench options are written in
 - `leveldb/benchmarks/db_bench.cc`
- Actual Benchmarks
 - `fillseq`
 - `fillrandom`
 - `readrandom`
 - `readseq`
- Meta operations
 - `compact`
 - `stats`
 - `sstables`

`leveldb / benchmarks / db_bench.cc`

```
// Comma-separated list of operations to run in the specified order
// Actual benchmarks:
// fillseq      -- write N values in sequential key order in async mode
// fillrandom   -- write N values in random key order in async mode
// overwrite    -- overwrite N values in random key order in async mode
// fillsync     -- write N/100 values in random key order in sync mode
// fill100K     -- write N/1000 100K values in random order in async mode
// deleteseq    -- delete N keys in sequential order
// deleterandom -- delete N keys in random order
// readseq      -- read N times sequentially
// readreverse  -- read N times in reverse order
// readrandom   -- read N times in random order
// readmissing  -- read N missing keys in random order
// readhot      -- read N times in random order from 1% section of DB
// seekrandom   -- N random seeks
// seekordered  -- N ordered seeks
// open         -- cost of opening a DB
// crc32c       -- repeated crc32c of 4K of data
// Meta operations:
// compact      -- Compact the entire DB
// stats        -- Print DB stats
// sstables     -- Print sstable info
// heapprofile  -- Dump a heap profile (if supported by this port)
```

Options

- All db_bench options are written in
 - leveldb/benchmarks/db_bench.cc

- Options

--histogram: latency histogram

--db: directory of db

--use_existing_db: true/false

--num: num of key-value pairs

--read: num of reads

--value_size(byte): size of value

--threads: num of threads

--cache_size(byte): size of block cache

--bloom_bits(bits): size of bloom filter

--max_file_size: max size of SSTable

```
65 // Number of key/values to place in database
66 static int FLAGS_num = 1000000;
67
68 // Number of read operations to do. If negative, do FLAGS_num reads.
69 static int FLAGS_reads = -1;
70
71 // Number of concurrent threads to run.
72 static int FLAGS_threads = 1;
73
74 // Size of each value
75 static int FLAGS_value_size = 100;
76
77 // Arrange to generate values that shrink to this fraction of
78 // their original size after compression
79 static double FLAGS_compression_ratio = 0.5;
80
81 // Print histogram of operation timings
82 static bool FLAGS_histogram = false;
83
84 // Count the number of string comparisons performed
85 static bool FLAGS_comparisons = false;
86
87 // Number of bytes to buffer in memtable before compacting
88 // (initialized to default value by "main")
89 static int FLAGS_write_buffer_size = 0;
90
91 // Number of bytes written to each file.
92 // (initialized to default value by "main")
93 static int FLAGS_max_file_size = 0;
94
95 // Approximate size of user data packed per block (before compression).
96 // (initialized to default value by "main")
97 static int FLAGS_block_size = 0;
98
99 // Number of bytes to use as a cache of uncompressed data.
100 // Negative means use default settings.
101 static int FLAGS_cache_size = -1;
102
103 // Maximum number of files to keep open at the same time (use default if == 0)
104 static int FLAGS_open_files = 0;
105
106 // Bloom filter bits per key.
107 // Negative means use default settings.
108 static int FLAGS_bloom_bits = -1;
109
110 // Common key prefix length.
111 static int FLAGS_key_prefix = 0;
112
113 // If true, do not destroy the existing database. If you set this
114 // flag and also specify a benchmark that wants a fresh database, that
115 // benchmark will fail.
116 static bool FLAGS_use_existing_db = false;
117
118 // If true, reuse existing log/MANIFEST files when re-opening a database.
119 static bool FLAGS_reuse_logs = false;
120
121 // Use the db with the following name.
122 static const char* FLAGS_db = nullptr;
```

```
1021 int main(int argc, char** argv) {
1022     FLAGS_write_buffer_size = leveldb::Options().write_buffer_size;
1023     FLAGS_max_file_size = leveldb::Options().max_file_size;
1024     FLAGS_block_size = leveldb::Options().block_size;
1025     FLAGS_open_files = leveldb::Options().max_open_files;
1026     std::string default_db_path;
1027
1028     for (int i = 1; i < argc; i++) {
1029         double d;
1030         int n;
1031         char junk;
1032         if (leveldb::Slice(argv[i]).starts_with("--benchmarks=")) {
1033             FLAGS_benchmarks = argv[i] + strlen("--benchmarks=");
1034         } else if (sscanf(argv[i], "--compression_ratio=%lf%c", &d, &junk) == 1) {
1035             FLAGS_compression_ratio = d;
1036         } else if (sscanf(argv[i], "--histogram=%d%c", &n, &junk) == 1 &&
1037             (n == 0 || n == 1)) {
1038             FLAGS_histogram = n;
1039         } else if (sscanf(argv[i], "--comparisons=%d%c", &n, &junk) == 1 &&
1040             (n == 0 || n == 1)) {
1041             FLAGS_comparisons = n;
1042         } else if (sscanf(argv[i], "--use_existing_db=%d%c", &n, &junk) == 1 &&
1043             (n == 0 || n == 1)) {
1044             FLAGS_use_existing_db = n;
1045         } else if (sscanf(argv[i], "--reuse_logs=%d%c", &n, &junk) == 1 &&
1046             (n == 0 || n == 1)) {
1047             FLAGS_reuse_logs = n;
1048         } else if (sscanf(argv[i], "--num=%d%c", &n, &junk) == 1) {
1049             FLAGS_num = n;
1050         } else if (sscanf(argv[i], "--reads=%d%c", &n, &junk) == 1) {
1051             FLAGS_reads = n;
1052         } else if (sscanf(argv[i], "--threads=%d%c", &n, &junk) == 1) {
1053             FLAGS_threads = n;
1054         } else if (sscanf(argv[i], "--value_size=%d%c", &n, &junk) == 1) {
1055             FLAGS_value_size = n;
1056         } else if (sscanf(argv[i], "--write_buffer_size=%d%c", &n, &junk) == 1) {
1057             FLAGS_write_buffer_size = n;
1058         } else if (sscanf(argv[i], "--max_file_size=%d%c", &n, &junk) == 1) {
1059             FLAGS_max_file_size = n;
1060         } else if (sscanf(argv[i], "--block_size=%d%c", &n, &junk) == 1) {
1061             FLAGS_block_size = n;
1062         } else if (sscanf(argv[i], "--key_prefix=%d%c", &n, &junk) == 1) {
1063             FLAGS_key_prefix = n;
1064         } else if (sscanf(argv[i], "--cache_size=%d%c", &n, &junk) == 1) {
1065             FLAGS_cache_size = n;
1066         } else if (sscanf(argv[i], "--bloom_bits=%d%c", &n, &junk) == 1) {
1067             FLAGS_bloom_bits = n;
1068         } else if (sscanf(argv[i], "--open_files=%d%c", &n, &junk) == 1) {
1069             FLAGS_open_files = n;
1070         } else if (strncmp(argv[i], "--db=", 5) == 0) {
1071             FLAGS_db = argv[i] + 5;
```

Metrics

- Default
 - Throughput (micros/op, MB/s)
- --histogram
 - Latency
- --benchmarks="stats"
 - Stats of SSTs
 - Num of file, Size of db / read /write
- --benchmarks="**any_bench**,stats,compact,stats"
 - SAF

```
mingu@sever:~/leveldb_release/build$ sudo su
[sudo] password for mingu:
root@sever:/home/mingu/leveldb_release/build# sh bench_script
./db_bench --use_existing_db=0 --compression_ratio=1 --comparis

LevelDB:    version 1.23
Date:       Sat Jul  9 16:36:24 2022
CPU:        16 * Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
CPUCache:   16384 KB
Keys:       16 bytes each
Values:     100 bytes each (100 bytes after compression)
Entries:    500000
RawSize:    55.3 MB (estimated)
FileSize:   55.3 MB (estimated)

-----
fillrandom  :      2.006 micros/op;   55.2 MB/s
Microseconds per op:
Count: 500000 Average: 2.0056 StdDev: 24.67
Min: 0.0000 Median: 1.8774 Max: 5425.0000

-----
[      0,      1 )      13   0.003%   0.003%
[      1,      2 ) 284903 56.981% 56.983% #####
[      2,      3 ) 193588 38.718% 95.701% #####
[      3,      4 )  17748  3.550% 99.250% #
[      4,      5 )   2395  0.479% 99.729%
[      5,      6 )    280  0.056% 99.785%
[      6,      7 )    110  0.022% 99.807%
```

```
Comparisons: 23384945

                                Compactions
Level  Files  Size(MB)  Time(sec)  Read(MB)  Write(MB)
-----
  0      0      0         0         0         47
  1      0      0         0         60        53
  2     19     34         1         95        88
```

Shell script

- Run db_bench with shell script
 - \$ sh bench_script.sh
- Contents
 - Environment setup
 - Option
 - Sample command
 - Record result
 - echo | tee -a result.txt
- If/For statement

```
# -----3. Run db_bench-----  
# clearing kernel buffers before running each workload.  
sync; echo 3 > /proc/sys/vm/drop_caches  
  
# sample db_bench command  
CMD="./db_bench \  
--use_existing_db=0 \  
--histogram=1 \  
--compression_ratio=1 \  
--comparisons=1 \  
--benchmarks="fillrandom,stats,readrandom,stats" \  
--num=500000 \  
--reads=300000 \  
--bloom_bits=0 \  
"  
  
echo "$CMD" | tee -a result.txt  
echo | tee -a result.txt  
  
RESULT=$( $CMD )  
  
echo "$RESULT" | tee -a result.txt  
echo | tee -a result.txt  
# -----
```


1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- db_bench experiment
 - ✓ Choose your topic
 - ✓ options / benchmarks / metrics
 - ✓ 5 steps of experiments

Choose your topic

- 6 Topics, 6 Teams
 - Memtable
 - WAL/Manifest
 - Compaction
 - SSTable
 - Bloom Filter
 - Cache
- Submit topics what you interested in
 - Deadline: 7/13, 11:00AM
 - <https://github.com/DKU-StarLab/leveldb-study/issues/3>

Options, Benchmarks, Metrics

LevelDB			
	Options	Benchmarks	Metrics
Memtable	write_buffer_size max_file_size	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF
SSTable	max_file_size block_size	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF
BloomFilter	bloom_bits (On/Off)	Readmissing Readrandom Seekrandom	Throughput Latency WAF, SAF
Cache	cache_size block_size	Readhot Readseq Readrandom Seekrandom	Throughput Latency RAF

Options, Benchmarks, Metrics

RocksDB			
	Options	Benchmarks	Metrics
Compaction	-base_background_compactions -compaction_style (level-based, universal, fifo)	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF
WAL	- disable_wal - wal_bytes_per_sync	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF

5 steps of experiment

1. hypotheses

- If option changes, what changes will happen internally?
- How will internal changes affect the metrics?
 - What result/graph do you expect?
- Why???

5 steps of experiment

2. Design

- Do the simplest and smallest experiment that can test your hypothesis.
- Do not experiment with multiple variables at once from the beginning
- Don't let uncontrolled variables ruin your experiment.

5 steps of experiment

2. Design

- Independent variable
 - Num of kv pairs
 - Options and benchmarks
- Dependent variable
 - Throughput, Latency, WAF/RAF ...
 - DB size(SAF)
- Controlled variable
 - Initial OS cache, other processes
 - Compile option (-g, -pg)
 - Default options (compression, bloom filter, cache, num, reads, using existing db)

5 steps of experiment

3. Run Experiment

- Use shell script or python script
 - Use echo for check your script is running correctly
- Use redirection to record result from shell
- Do not use leveldb which complied with debug/profile options
- Turn off other processes

5 steps of experiment

4. Result and Discussion

- Verify your idea/hypothesis with result data.
- Explain why your hypothesis is correct or not.

5 steps of experiment

5. Present your experiment in 10 minutes

- 1. Hypotheses
- 2. Design
- 3. Environment
- 4. Result and Discussion
 - Extra 5 min for Discussion

■ Previous study presentation example

- https://github.com/DKU-StarLab/RocksDB_Festival

Notice

- Upload your presentation file through pull request
 - Make pull request until 7/19 11AM
 - `leveldb-study/analysis/benchmark/`
- PPT format is uploaded in github introduction folder
 - Format: `[your topic] benchmark analysis.pdf`
 - <https://github.com/DKU-StarLab/leveldb-study/tree/main/Introduction>
- The week after next
 - write a document that explains your experiment
 - upload in git-book

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- References
 - ✓ Documents
 - ✓ Lectures

Documents

leveldb

Jeff Dean, Sanjay Ghemawat

The leveldb library provides a persistent key value store. Keys and values are arbitrary byte arrays. The keys are ordered within the key value store according to a user-specified comparator function.

Opening A Database

A leveldb database has a name which corresponds to a file system directory. All of the contents of database are stored in this directory. The following example shows how to open a database, creating it if necessary:

```
#include <cassert>
#include "leveldb/db.h"

leveldb::DB* db;
leveldb::Options options;
options.create_if_missing = true;
leveldb::Status status = leveldb::DB::Open(options, "/tmp/testdb", &db);
assert(status.ok());
...
```

If you want to raise an error if the database already exists, add the following line before the `leveldb::DB::Open` call:

leveldb/doc/

facebook / rocksdbPublic

Watch1kFork5.3kStar22.9k

<> CodeIssues425Pull requests270ActionsProjectsWikiSecurityInsights

Home

cheng-chang edited this page on 19 Jan · 57 revisions

Welcome to RocksDB

Pages156

RocksDB is a storage engine with key/value interface, where keys and values are arbitrary byte streams. It is a C++ library. It was developed at Facebook based on LevelDB and provides backwards-compatible support for LevelDB APIs.

RocksDB supports various storage hardware, with fast flash as the initial focus. It uses a Log Structured Database Engine for storage, is written entirely in C++, and has a Java wrapper called RocksJava. See [RocksJava Basics](#).

RocksDB can adapt to a variety of production environments, including pure memory, Flash, hard disks or remote storage. Where RocksDB cannot automatically adapt, highly flexible configuration settings are provided to allow users to tune it for them. It supports various compression algorithms and good tools for production support and debugging.

Features

- Designed for application servers wanting to store up to a few terabytes of data on local or remote storage systems.
- Optimized for storing small to medium size key-values on fast storage -- flash devices or in-memory
- It works well on processors with many cores

Features Not in LevelDB

Contents

- RocksDB Wiki
- Overview
- RocksDB FAQ
- Terminology
- Requirements
- Contributors' Guide
- Release Methodology
- RocksDB Users and Use Cases
- RocksDB Public Communication and Information Channels
- Basic Operations
 - Iterator
 - Prefix seek
 - SeekForPrev
 - Tailing Iterator
 - Compaction Filter
 - Read-Modify-Write (Merge) Operator
 - Column Families
 - Creation and Ingestion SST files

rocksdb github wiki

Documents

leveldb-handbook

latest

Search docs

内容:

基本概念

读写操作

日志

内存数据库

sstable

缓存系统

布隆过滤器

compaction

版本控制

DigitalOcean

Digital Ocean: Create your world-changing apps on the cloud developers love Try now with a \$100 Credit

Ad by EthicalAds · Monetize your site

Docs » leveldb-handbook

leveldb-handbook

内容:

- 基本概念
 - 整体架构
- 读写操作
 - 写操作
 - 读操作
 - 读取
- 日志
 - 日志结构
 - 日志内容
 - 日志写
 - 日志读
- 内存数据库
 - 跳表
 - 内存数据库
- sstable
 - 概述
 - SStable文件格式
 - data block结构
 - filter block结构

<https://leveldb-handbook.readthedocs.io/zh/latest/>

rsy56640 / read_and_analyse_levelDB Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights

master read_and_analyse_levelDB / reference / Go to file

rsy56640 Update reference 76e3f1

..

DB leveldb实现解析.pdf ..

LevelDB源码分析.docx ..

LevelDB源码分析.pdf ..

README.md Update reference

bigtable-osdi06.pdf ..

leveldb-handbook.pdf ..

lsmtree.pdf ..

README.md

Reference

- leveldb实现解析 - 淘宝-核心系统研发-存储
- leveldb-handbook
- LevelDB: Read the Fucking Source Code.
- leveldb - 超全讲解.....
- LevelDB源码分析 - 百度文库 100多页.....
- The principle of LevelDb analysis - 英文，很多文章的出处
- Leveldb代码阅读笔记 - codedump
- 存储引擎技术架构与内幕 (leveldb-1) #58

rsy56640/read_and_analyse_levelDB

Documents

LevelDB Introduction

Fenggang Wu
Oct. 17th, 2016

Fenggang Wu, 『LevelDB Introduction』, University of Minnesota CSci5980, 2016

DKU-StarLab / RocksDB_Festival Public

Dankook University RocksDB research/study

☆ 12 stars 🍴 8 forks

★ Starred

👁 Watch

Code

Issues

Pull requests

Actions

Projects

...

https://github.com/DKU-StarLab/RocksDB_Festival

Lectures

:D-MOOC

이용안내

콘텐츠 검색

공지사항

Q&A

연관 사이트

로그인

상시 접수

비정형 빅데이터를 위한 키-밸류 DB

비정형 빅데이터를 위한 키-밸류 DB

최종무 | 상시모집 | 학습 기간 2021.12.01 ~ 2030.12.31

강좌 언어	강의 길이	권장 학습 시간	이수증 제공 여부
한국어	7 주	4 시간	미제공

신청현황 30 명 / 무제한

강좌소개

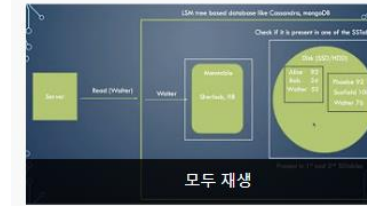
키-밸류 DB는 최근 주목받고 있는 비정형 빅데이터를 위한 데이터베이스임.

대표적인 예로 구글의 LevelDB, 페이스북의 RocksDB, 아마존의 Dynamo 등이 있음.

신청 기간
상시모집

제공 기관

[Jongmoo Choi, 『Key-Value DB for Unstructured data』, 2021](#)



LSM Trees

동영상 3개 · 조회수 68회 · 최종 업데이트: 2021. 7. 27.

GL Tech Tutorials



GL Tech Tutorials

구독

- 1 LSM trees (Log Structured Merge Trees) - Detailed video
GL Tech Tutorials
19:12
- 2 LSM trees (Log Structured Merge Trees) - Detailed write path | Part 1
GL Tech Tutorials
11:48
- 3 Bloom Filters in LSM Trees | Counting Bloom Filters | System Design
GL Tech Tutorials
14:27

[GL Tech Tutorials, 『LSM trees』, 2021](#)



leveldb

동영상 18개 · 조회수 2,542회 · 최종 업데이트: 2021. 11. 21.

Wei Zhou



Wei Zhou

구독

사용할 수 없는 동영상 1개가 숨겨졌습니다.

- 1 Scaling concurrent log-structured data stores
Association for Computing Machinery (ACM)
20:36
- 2 Intro to LevelDB
Kyle Robinson Young
15:40
- 3 Jeff Dean: "Achieving Rapid Response Times in Large Online Services" Keynote - Velocity 2014
O'Reilly
28:00
- 4 Learning LevelDB with Julian Gruber and Zeke Sikelianos
Zeke Sikelianos
41:30

[Wei Zhou, LevelDB YouTube playlist](#)

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

■ Homework

✓ Submit your topic until 7/13 11AM

- Your team will be announced at 7/13 2PM

✓ Experiment and presentation

- Install Level DB
- Run Experiment
- Prepare presentation
- Pull request your presentation pdf file until 7/19 11AM

1. LevelDB Architecture
2. Key-Value Interface
3. Internal Operations
4. Data Structure
5. LevelDB Installation
6. db_bench experiment
7. References
8. Homework

- Next week
 - ✓ Student presentation
 - Experiment
 - ✓ Lecture
 - How to analyze LevelDB

Thank you

Appendix

- How to install RocksDB

- Install guide

- <https://github.com/facebook/rocksdb/blob/main/INSTALL.md>

```
$ sudo apt-get update
```

```
$ sudo apt-get install build-essential
```

```
$ sudo apt-get install cmake
```

```
$ sudo apt-get install libgflags-dev
```

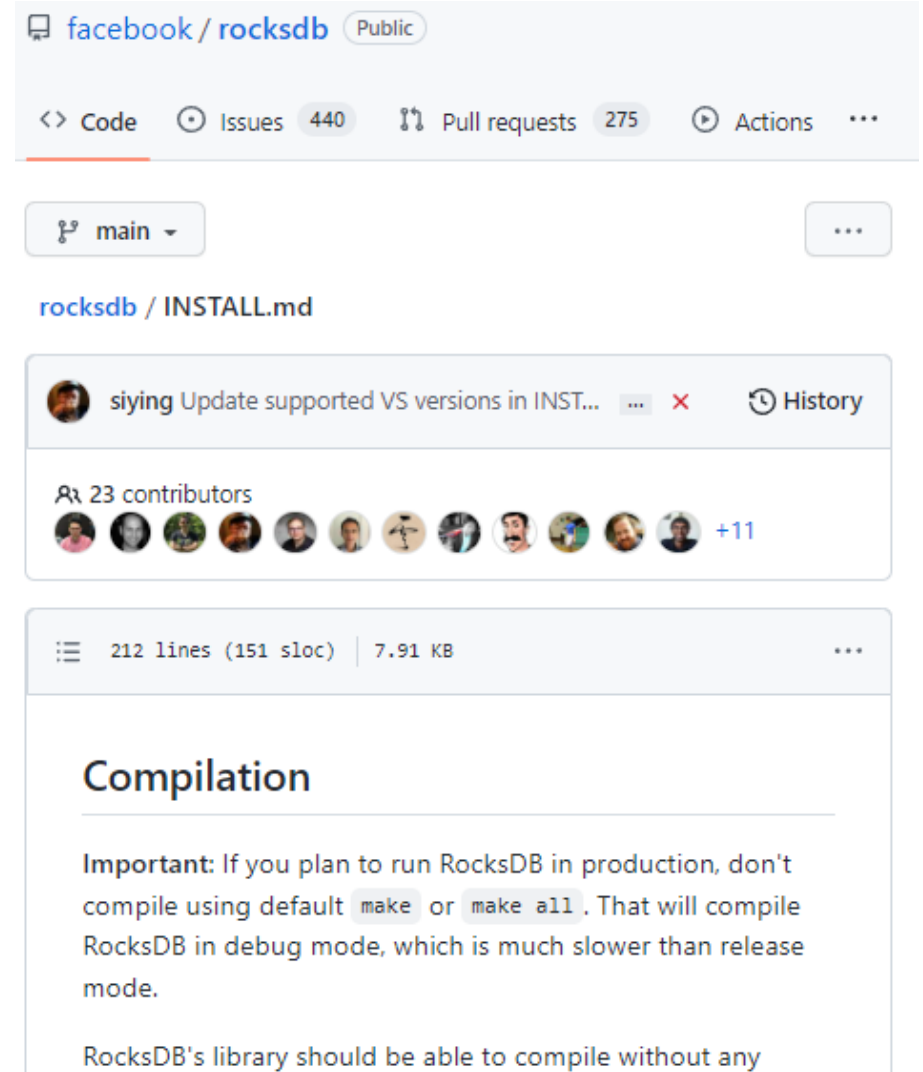
```
$ sudo apt-get install libsnappy-dev
```

```
$ git clone https://github.com/facebook/rocksdb.git
```

```
$ cd rocksdb
```

```
$ make db_bench -j (cpu_core)
```

```
$ db_bench
```



The screenshot displays the GitHub interface for the `facebook/rocksdb` repository. At the top, the repository name and "Public" status are shown. Below this, navigation links for "Code", "Issues" (440), "Pull requests" (275), and "Actions" are visible. A dropdown menu shows the current branch as "main". The file path `rocksdb / INSTALL.md` is highlighted. A commit message by user "siying" is shown: "Update supported VS versions in INST...". Below the commit, a row of 23 contributor avatars is displayed. The file statistics indicate 212 lines (151 sloc) and 7.91 KB. The "Compilation" section contains an important note: "Important: If you plan to run RocksDB in production, don't compile using default `make` or `make all`. That will compile RocksDB in debug mode, which is much slower than release mode." It also states that "RocksDB's library should be able to compile without any".

<https://github.com/facebook/rocksdb/blob/main/INSTALL.md>

Appendix

■ RocksDB db_bench option

Benchmarking tools

Ting Sun edited this page on 11 Feb · 6 revisions

db_bench

`db_bench` is the main tool that is used to benchmark RocksDB's performance. RocksDB inherited `db_bench` from LevelDB, and enhanced it to support many additional options. `db_bench` supports many benchmarks to generate different types of workloads, and its various options can be used to control the tests.

If you are just getting started with `db_bench`, here are a few things you can try:

1. Start with a simple benchmark like `fillseq` (or `fillrandom`) to create a database and fill it with some data

```
./db_bench --benchmarks="fillseq"
```

If you want more stats, add the meta operator "stats" and `--statistics` flag.

```
./db_bench --benchmarks="fillseq,stats" --statistics
```

2. Read the data back

```
./db_bench --benchmarks="readrandom" --use_existing_db
```

You can also combine multiple benchmarks to the string that is passed to `--benchmarks` so that they run sequentially. Example:

```
./db_bench --benchmarks="fillseq,readrandom,readseq"
```

More in-depth example of `db_bench` usage can be found [here](#) and [here](#).

Pages 156

Contents

- RocksDB Wiki
- Overview
- RocksDB FAQ
- Terminology
- Requirements
- Contributors' Guide
- Release Methodology
- RocksDB Users and Use Cases
- RocksDB Public Communication and Information Channels
- Basic Operations
 - Iterator
 - Prefix seek
 - SeekForPrev
 - Tailing Iterator
 - Compaction Filter
 - Read-Modify-Write (Merge) Operator
 - Column Families
 - Creating and Ingesting SST files
 - Single Delete
 - Low Priority Write
 - Time to Live (TTL) Support
 - Transactions
 - Snapshot
 - DeleteRange
 - Atomic flush

<https://github.com/facebook/rocksdb/wiki/Benchmarking-tools>

facebook / rocksdb Public Watch 1k Fork 5.3k Star 23k

<> Code Issues 440 Pull requests 275 Actions Projects Wiki ...

main rocksdb / tools / db_bench_tool.cc Go to file ...

mdcallag Set the value for --version, add --buil... ✓ Latest commit 177b2fa 4 days ago History

98 contributors +64

8567 lines (7708 sloc) 314 KB Raw Blame

```
1 // Copyright (c) 2011-present, Facebook, Inc. All rights reserved.
2 // This source code is licensed under both the GPLv2 (found in the
3 // COPYING file in the root directory) and Apache 2.0 License
4 // (found in the LICENSE.Apache file in the root directory).
5 //
```

https://github.com/facebook/rocksdb/blob/main/tools/db_bench_tool.cc

Appendix

RocksDB			
	Options	Benchmarks	Metrics
Compaction	-base_background_compactions -compaction_style (level-based, universal, fifo)	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF
WAL	- disable_wal - wal_bytes_per_sync	Fillseq Fillrandom Readrandom	Throughput Latency WAF, SAF