ProjectDb

A Key Value Storage Library

Lieyang Chen
Mengwen Li (ml4643)
Shengtan Mao (sm4954)

https://github.com/mli9502/ProjectDb

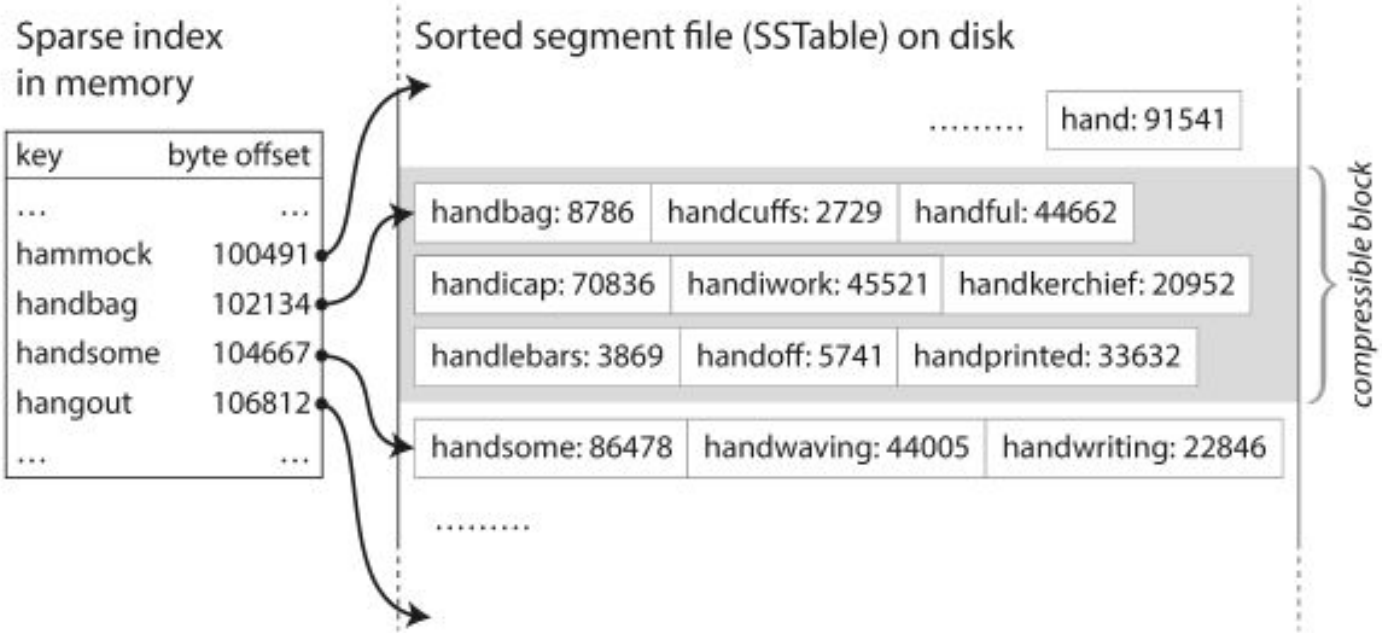
Introduction

- In this project, we implemented a key-value storage engine library by keeping record
- Provides a better performance comparing to directly writing file/reading file from disk.
- Provide a user friendly interface.
-
- Currently Distributed NoSQL database is very popular and widely used, things like Cassandra, MongoDb, … And a key-value storage engine could be used as a base for these distributed NoSQL databases.

Basic Terminologies

- MemTable: A table containing sorted key-value entries stored in memory.
- SSTable(aka segment file): A table containing sorted key-value entries stored in disk. (Created by flush Memtable into disk).
- Segment: A small-size block of sorted key-value entries stored in disk. Memory reads in a segment instead of entire SSTable when performing read operations. (SSTable is composed of many segments).
- Sparse Index Table: A table that keeps track of the position of the beginning of each Segment.

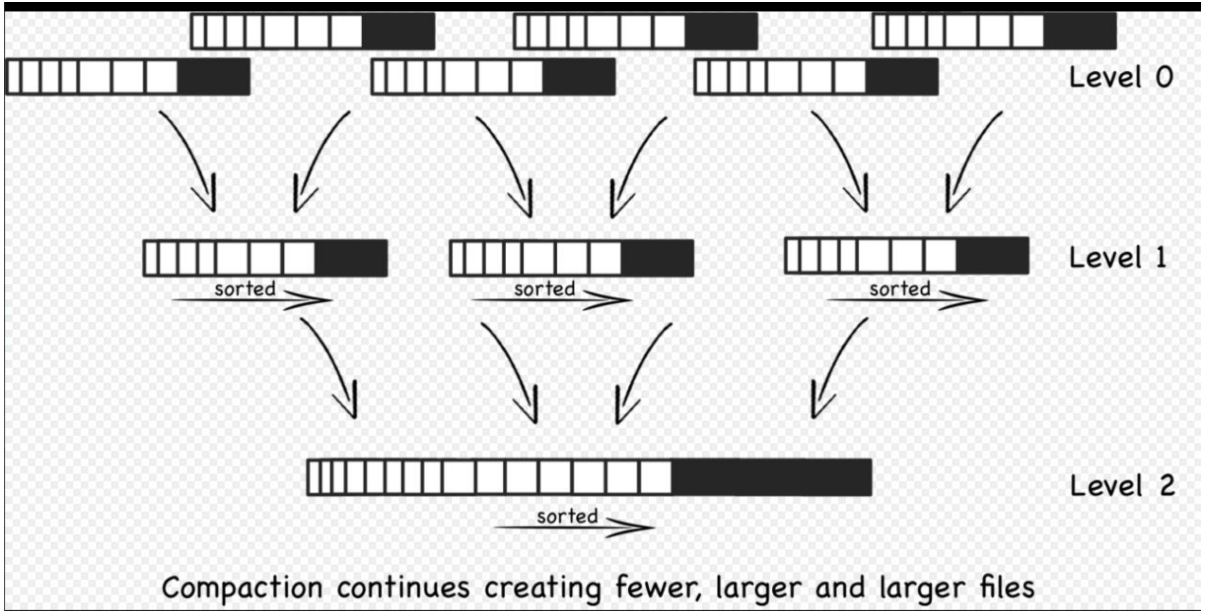
An intuitive Graph



source: Designing Data-Intensive Applications: p76 - p79

Algorithm

LSM-Tree(Log-structured merge-tree)


Level 0
Level 1
sorted
sorted
sorted
Level 2
sorted
Compaction continues creating fewer, larger and larger files

source: https://en.wikipedia.org/wiki/Log-structured_merge-tree#/media/File:LSM_Tree.png

Workflow

- When a write comes in, add it to an in-memory balanced tree data structure (for example, a red-black tree). This in-memory tree is sometimes called a **memtable**.
- When the memtable gets bigger than some threshold—typically a few megabytes—write it out to disk as an **SSTable** file. This can be done efficiently because the tree already maintains the key-value pairs sorted by key. The new SSTable file becomes the most recent segment of the database. While the SSTable is being written out to disk, writes can continue to a new memtable instance.
- In order to serve a read request, first try to find the key in the memtable, then in the most recent on-disk segment, then in the next-older segment, etc.
- From time to time, run a **merging** and compaction process in the background to combine segment files and to discard overwritten or deleted values.

Complexity

Append-only

Deletion is not decreasing file sizes

Complexity

Append-only

Deletion is not decreasing file sizes

Serialization/Deserialization

Append-only

Deletion is not decreasing file sizes

Serialization/Deserialization

Append-only

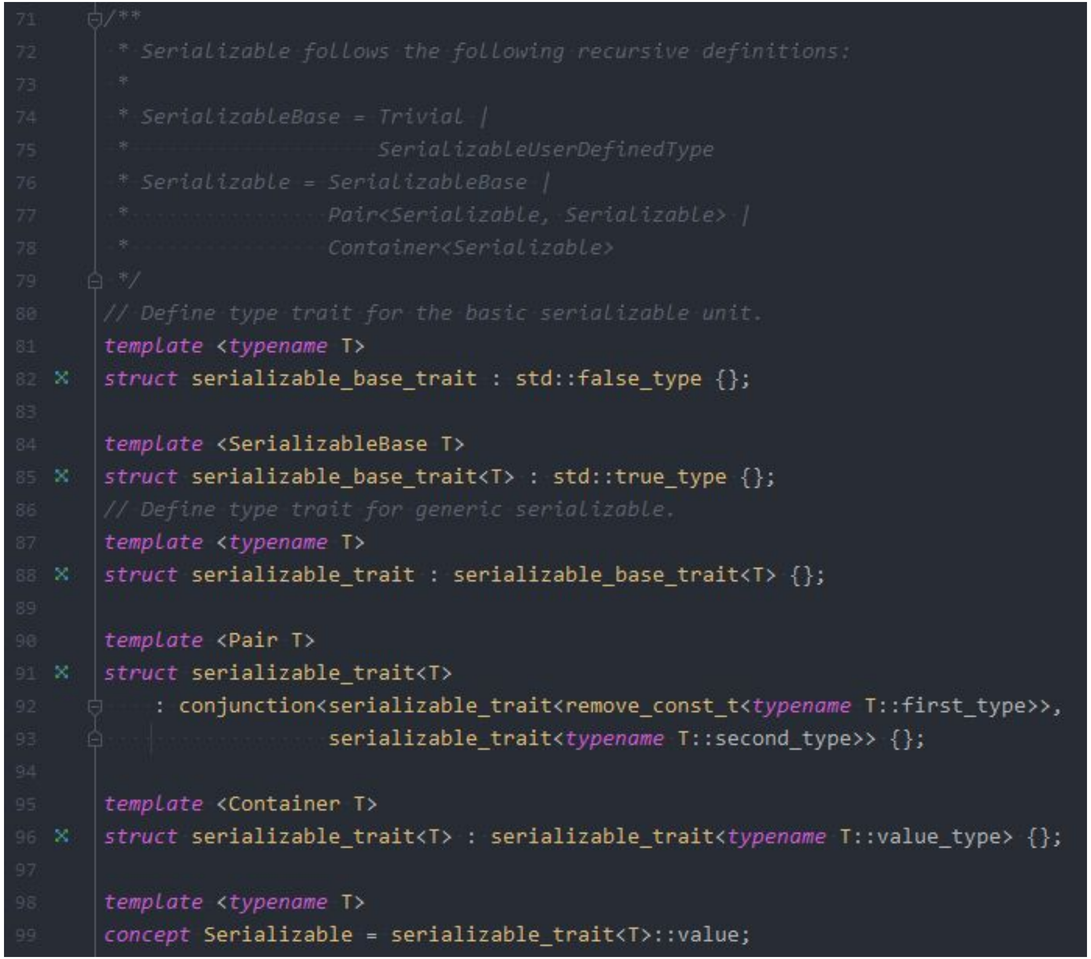
Deletion is not decreasing file sizes

Serialization/Deserialization

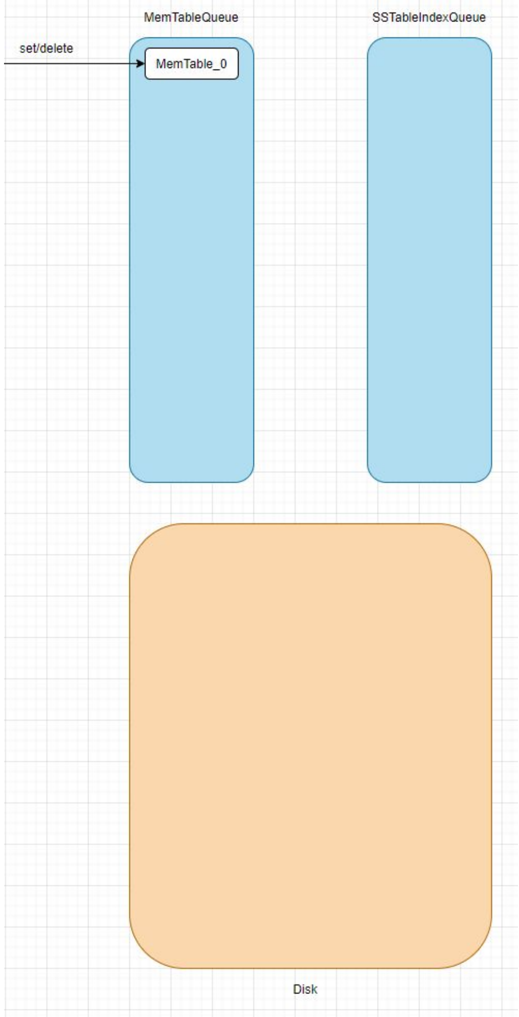
Append-only

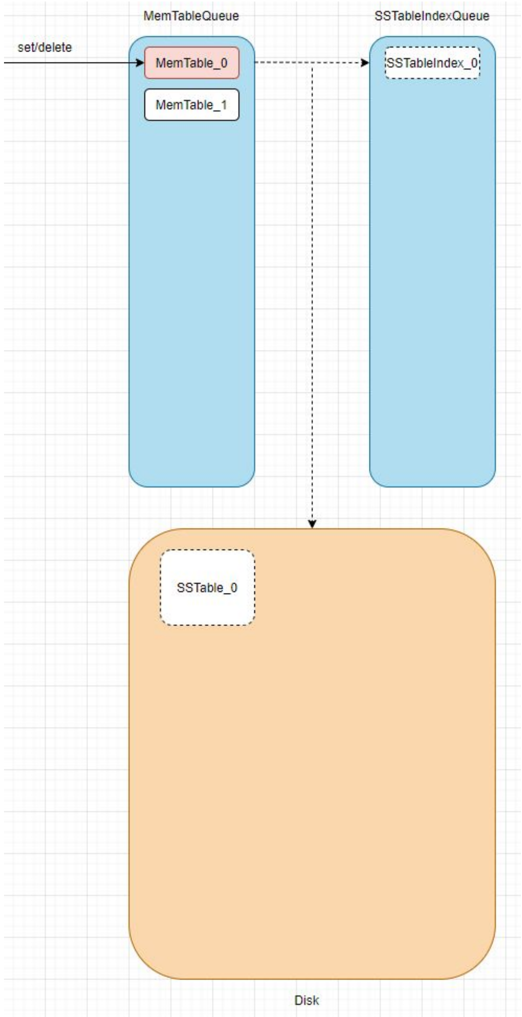
Deletion is not decreasing file sizes

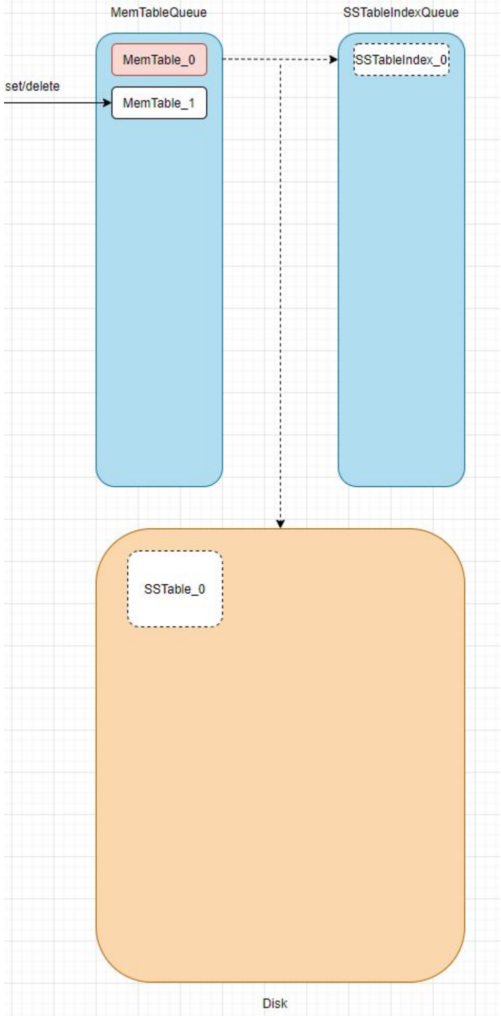
Serialization/Deserialization

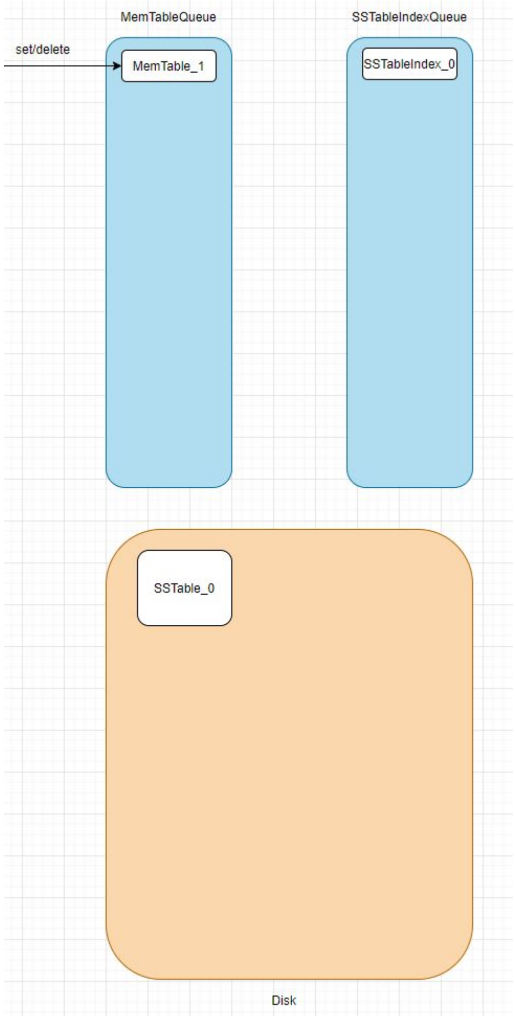


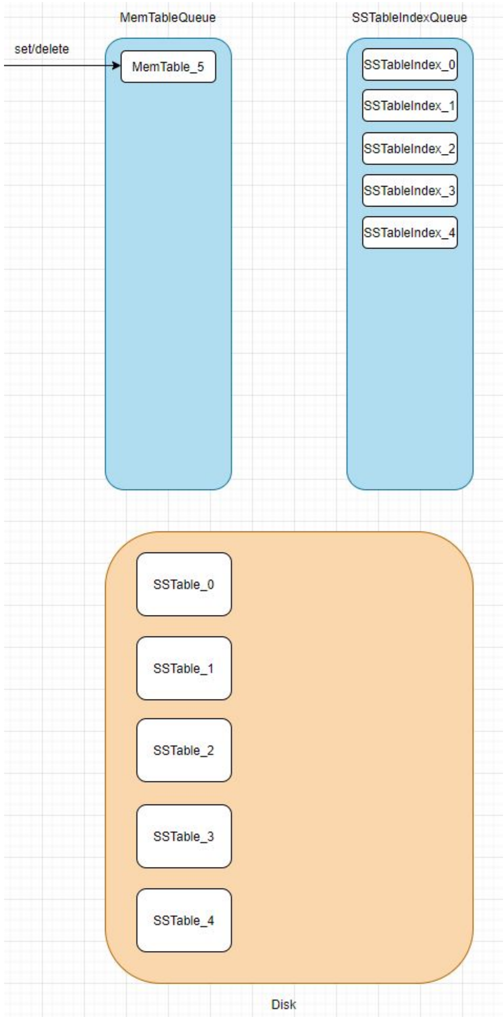
```cpp
/**
 * Serializable follows the following recursive definitions:
 *
 * SerializableBase = Trivial |
 *                    SerializableUserDefinedType
 * Serializable = SerializableBase |
 *                Pair<Serializable, Serializable> |
 *                Container<Serializable>
 */
// Define type trait for the basic serializable unit.
template <typename T>
struct serializable_base_trait : std::false_type {};

template <SerializableBase T>
struct serializable_base_trait<T> : std::true_type {};
// Define type trait for generic serializable.
template <typename T>
struct serializable_trait : serializable_base_trait<T> {};

template <Pair T>
struct serializable_trait<T>
    : conjunction<serializable_trait<remove_const_t<typename T::first_type>>,
                  serializable_trait<typename T::second_type>> {};

template <Container T>
struct serializable_trait<T> : serializable_trait<typename T::value_type> {};

template <typename T>
concept Serializable = serializable_trait<T>::value;
```

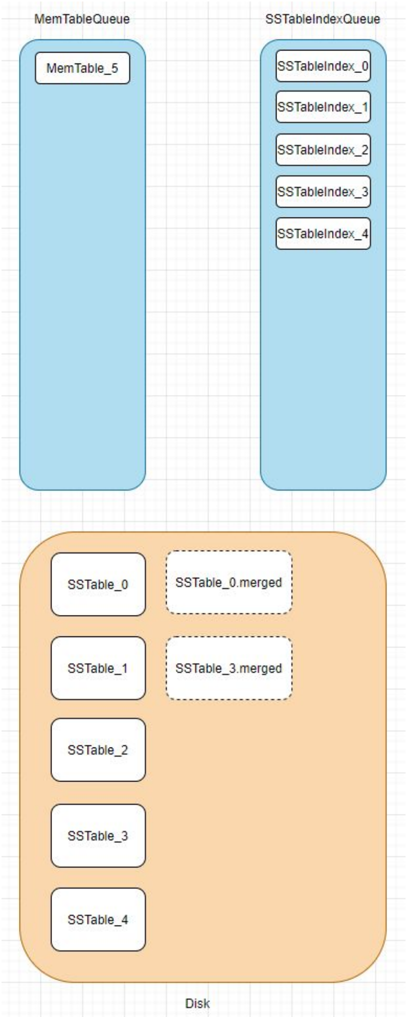

MemTableQueue
SSTableIndexQueue
set/delete
MemTable_0
Disk

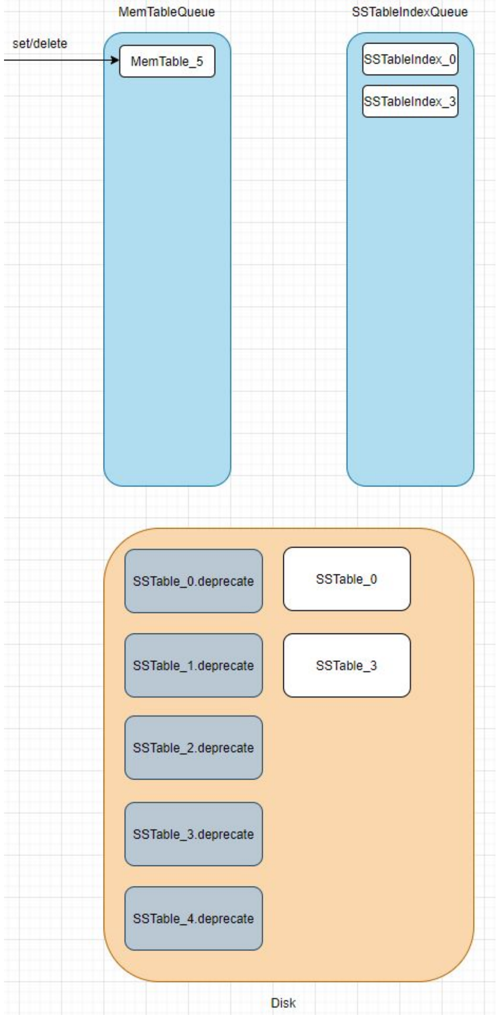

MemTableQueue
SSTableIndexQueue
set/delete
MemTable_0
MemTable_1
SSTableIndex_0
SSTable_0
Disk

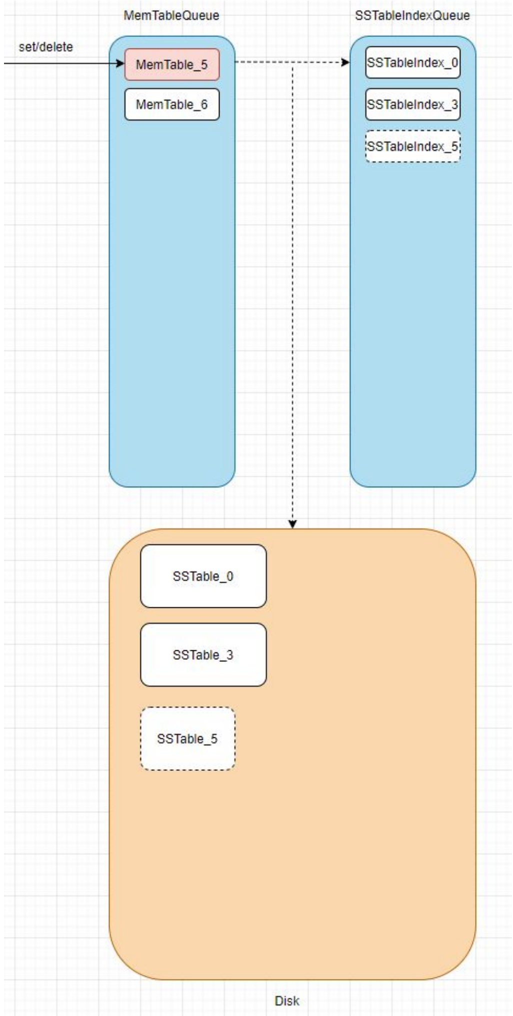

MemTableQueue
SSTableIndexQueue
MemTable_0
SSTableIndex_0
set/delete
MemTable_1
SSTable_0
Disk


MemTableQueue
SSTableIndexQueue
set/delete
MemTable_1
SSTableIndex_0
SSTable_0
Disk


MemTableQueue
SSTableIndexQueue
set/delete
MemTable_5
SSTableIndex_0
SSTableIndex_1
SSTableIndex_2
SSTableIndex_3
SSTableIndex_4
SSTable_0
SSTable_1
SSTable_2
SSTable_3
SSTable_4
Disk


MemTableQueue
SSTableIndexQueue
MemTable_5
SSTableIndex_0
SSTableIndex_1
SSTableIndex_2
SSTableIndex_3
SSTableIndex_4
SSTable_0
SSTable_0.merged
SSTable_1
SSTable_3.merged
SSTable_2
SSTable_3
SSTable_4
Disk


MemTableQueue
SSTableIndexQueue
set/delete
MemTable_5
SSTableIndex_0
SSTableIndex_3
SSTable_0.deprecate
SSTable_0
SSTable_1.deprecate
SSTable_3
SSTable_2.deprecate
SSTable_3.deprecate
SSTable_4.deprecate
Disk


MemTableQueue
SSTableIndexQueue
set/delete
MemTable_5
MemTable_6
SSTableIndex_0
SSTableIndex_3
SSTableIndex_5
SSTable_0
SSTable_3
SSTable_5
Disk

Measurements

Main features of Database:

writing - .set()

removing - .remove(), which works similarly to .set()

reading - .get()

Supports randomly generated data

Supports reading data from csv

Data for Benchmark

Key value pairs stored in a vector

Randomly generated

- 0, 1, 2, … as keys
- randomly generated string as values

Trending YouTube Video Statistics

- video ID as keys
- video title as values

Benchmark Description

7 Tasks:

- fill sequential / random
- overwrite
- delete sequential / random
- read sequential / random

Benchmark: Randomly Generated

First configuration

  2 mb in memory

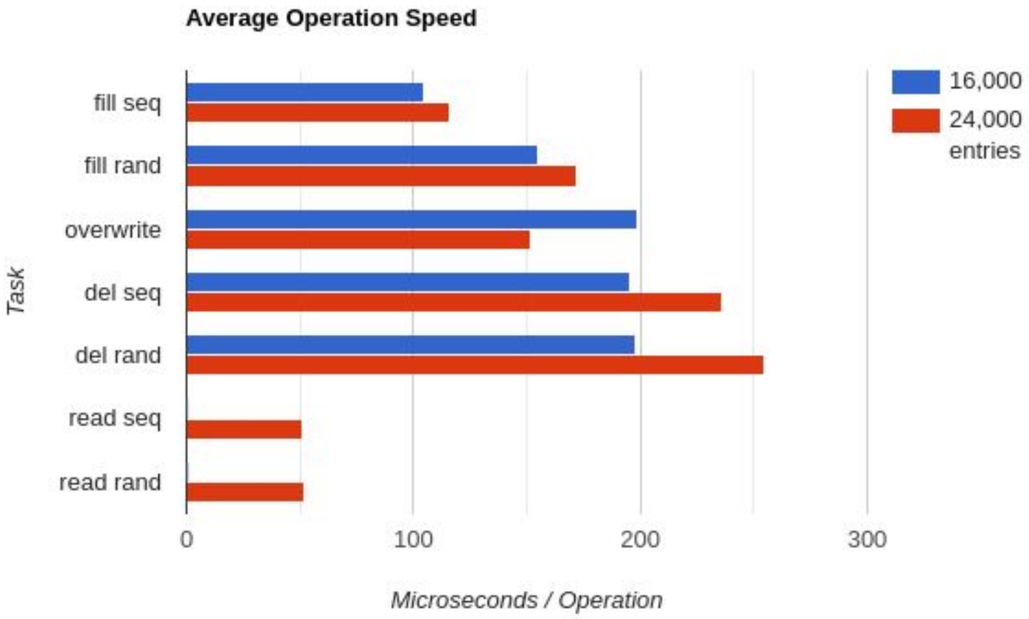
  250 kb index block

Slower for more entries

Fill faster than delete

Sequential faster than random

Overwrite?



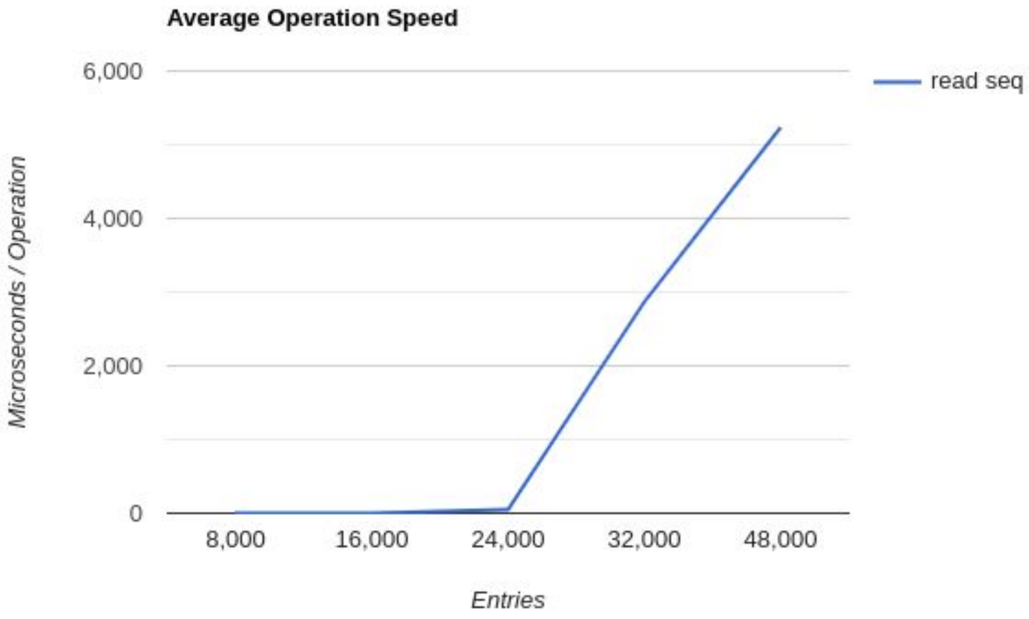
Benchmark: Read

Fast for low numbers

Slow when data size > 2 mb

values: 100 characters

2.4 mb for 24,000 entries



How to Solve This?

32,000 entries

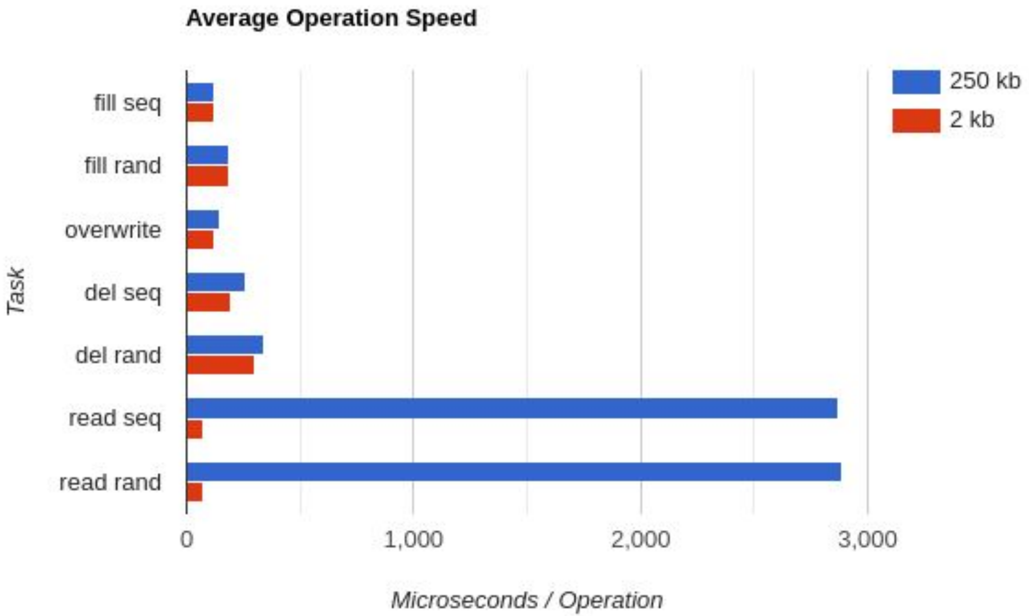
Second configuration

2 mb in memory

2 kb index block

Much faster read

Similar performance otherwise



Benchmark: Trending YouTube Video Statistics

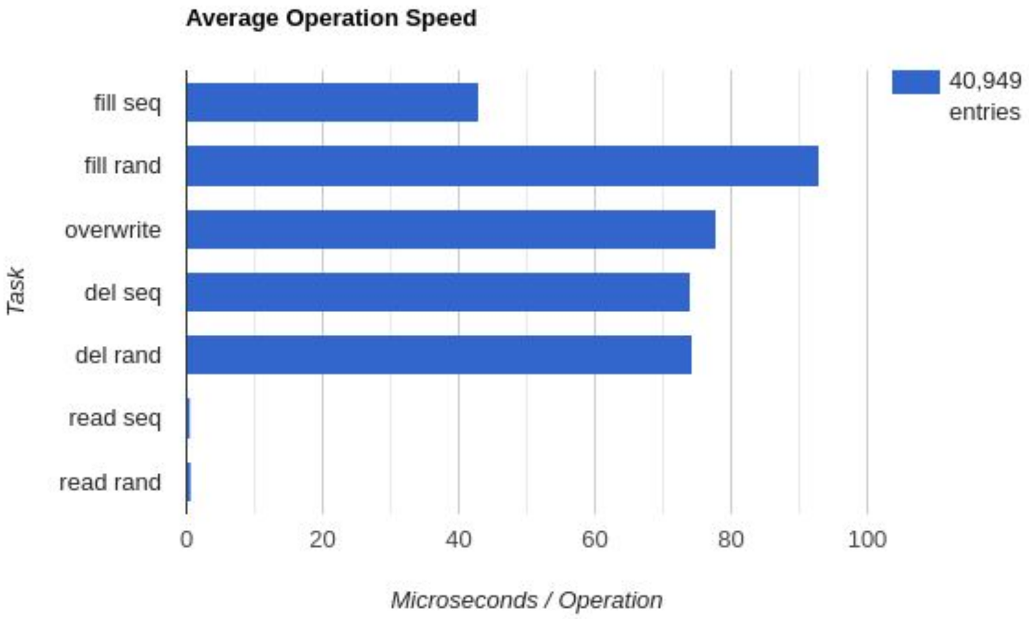
Example:

2kyS6SvSYSE

“WE WANT TO TALK ABOUT OUR MARRIAGE”

Using the first configuration

Smaller value sizes



Further Benchmarks

Comparison with LevelDB

Flush to disk

More configurations

Questions?

- Measurements
  - randomly generated and csv
  - writing, removing, reading
  - 2 kb index block faster