ProjectDb

A Key Value Storage Library

Lieyang Chen (lc3548)
Mengwen Li (ml4643)
Shengtan Mao (sm4954)

https://github.com/mli9502/ProjectDb

Acknowledgements and References

Martin Kleppmann: Designing Data-Intensive Applications

LevelDB: https://github.com/google/leveldb

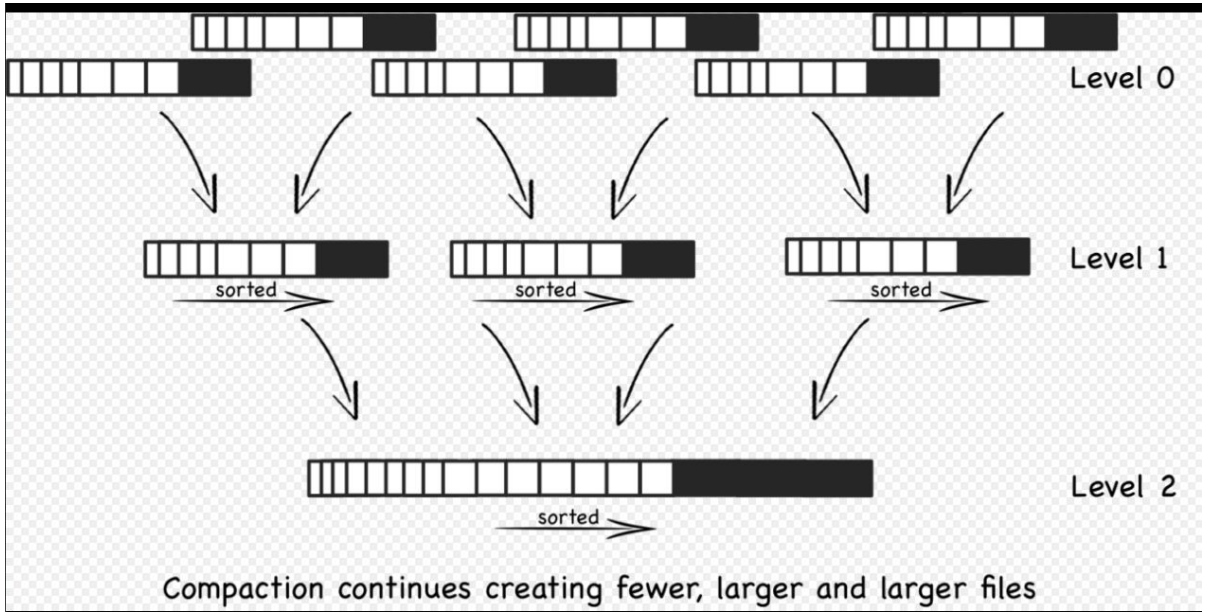
Trending YouTube Video Statistics:
https://www.kaggle.com/datasnaek/youtube-new?select=USvideos.csv

Introduction

- In this project, we implemented a key-value storage engine library
- Provides a better performance comparing to directly writing file/reading file from disk.
- Provide a user friendly interface.
- A key-value storage engine could be used as a base for distributed NoSQL databases.

Algorithm

LSM-Tree (Log-structured merge-tree)

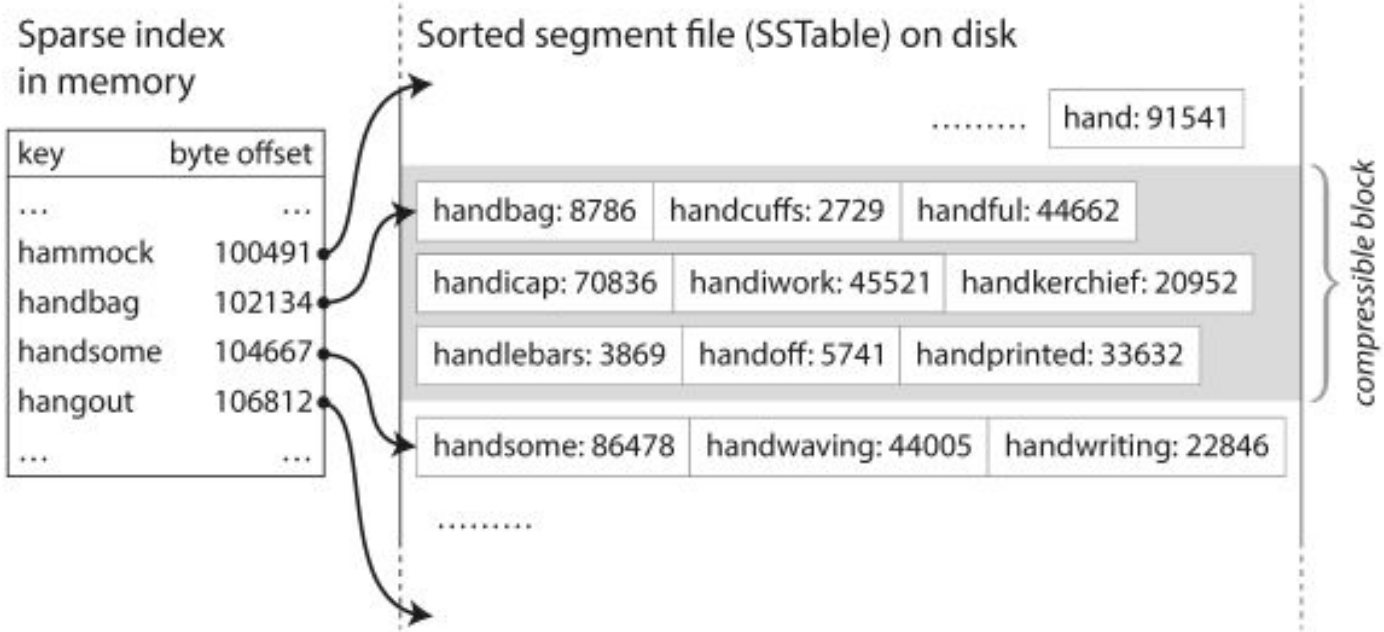


source: https://en.wikipedia.org/wiki/Log-structured_merge-tree#/media/File:LSM_Tree.png

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 (**SSTableIndex**): A table that keeps track of the position of the beginning of each Segment.

An intuitive Graph



source: Designing Data-Intensive Applications: p76 - p79

Workflow

- When a write comes in, add it to an in-memory table sorted by key. (MemTable)
- When MemTable gets bigger than some threshold, write it out to disk as an **SSTable** file.
- In order to serve a read request, first try to find the key in MemTable, then in the most recent on-disk **SSTable**, then in the next-older **SSTable**, 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.

APIs

- void set(const string& key, const string& value);
- string get(const string& key);
- string remove(const string& key);

Serialization/Deserialization

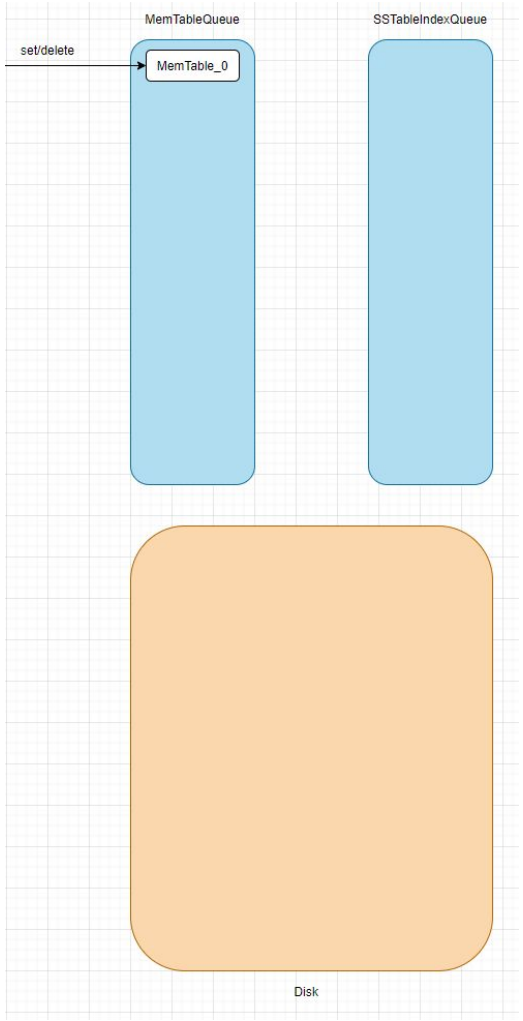
```
/**
 * Serializable follows the following recursive definitions:
 *
 * SerializableBase = Trivial |
 *                    SerializableUserDefinedType
 * Serializable = SerializableBase |
 *                Pair<Serializable, Serializable> |
 *                Container<Serializable>
 */
// Define type 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;
```

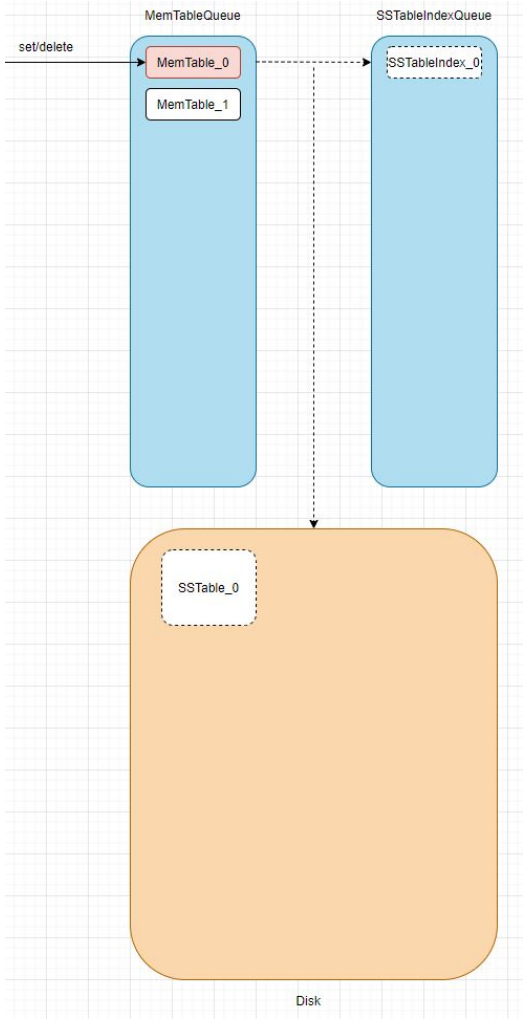
Serialization/Deserialization

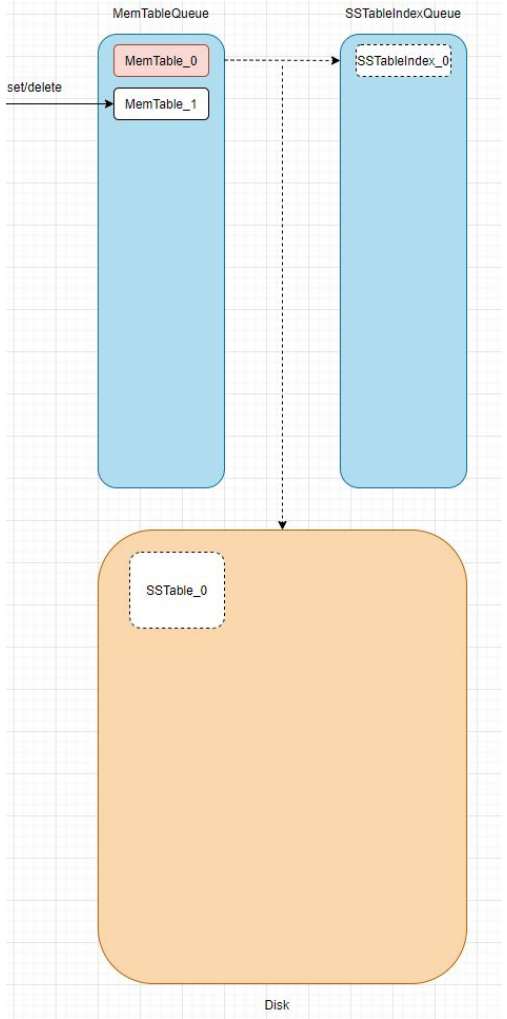
```
136  template <SerializableUserDefinedType T>
137  class SerializationWrapper<T> : public impl::SerializationWrapperBase<T> {
138      public:
139          explicit SerializationWrapper(const T& t)
140              : impl::SerializationWrapperBase<T>(t){};
141          explicit SerializationWrapper(T&& t)
142              : impl::SerializationWrapperBase<T>(move(t)) {}
143
144          void operator()(ostream& os) && {
145              const T& t = this->getCRefT();
146              t.serializeImpl(os);
147              if (!os) {
148                  log::errorAndThrow("Failed to serialize Serializable data!");
149              }
150          }
151  };
```

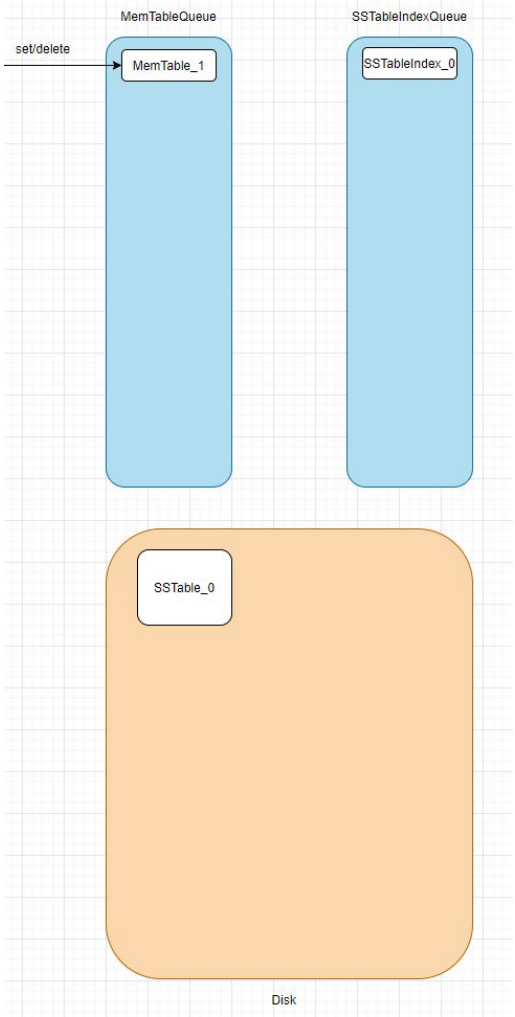
Serialization/Deserialization

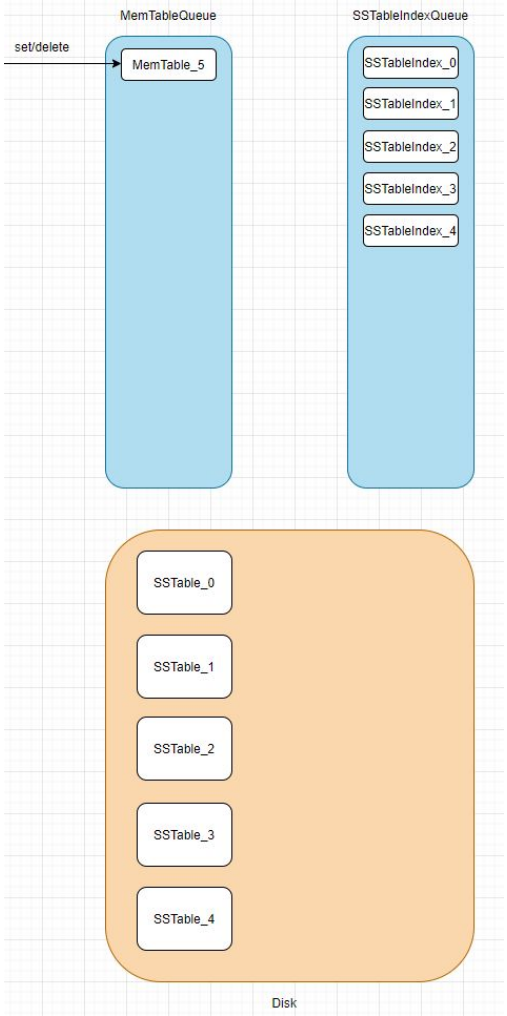
```
169  template <SerializablePair T>
170  class SerializationWrapper<T> : public impl::SerializationWrapperBase<T> {
171      public:
172          explicit SerializationWrapper(const T& t)
173              : impl::SerializationWrapperBase<T>(t){};
174          explicit SerializationWrapper(T&& t)
175              : impl::SerializationWrapperBase<T>(move(t)) {}
176
177          void operator()(ostream& os) && {
178              const T& t = this->getCRefT();
179              SerializationWrapper<
180                  typename remove_const<typename T::first_type>::type>(t.first)(os);
181              SerializationWrapper<typename T::second_type>(t.second)(os);
182          }
183  };
```

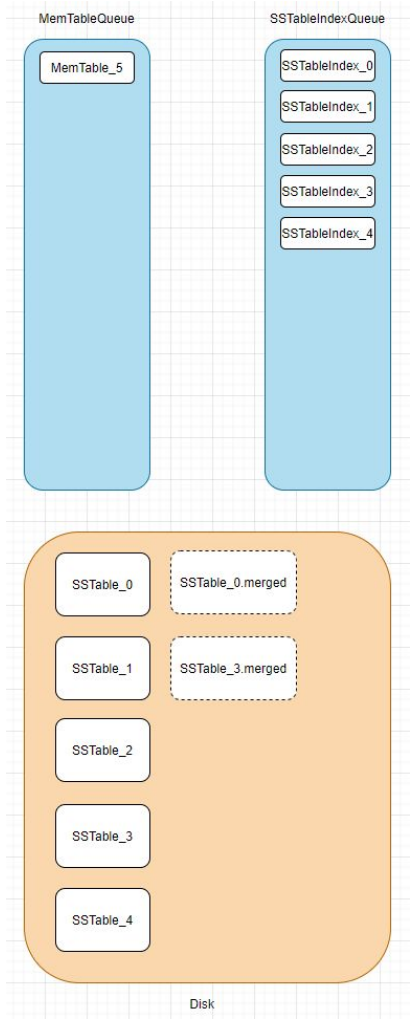

MemTableQueue
SSTableIndexQueue
set/delete
MemTable_0
Disk

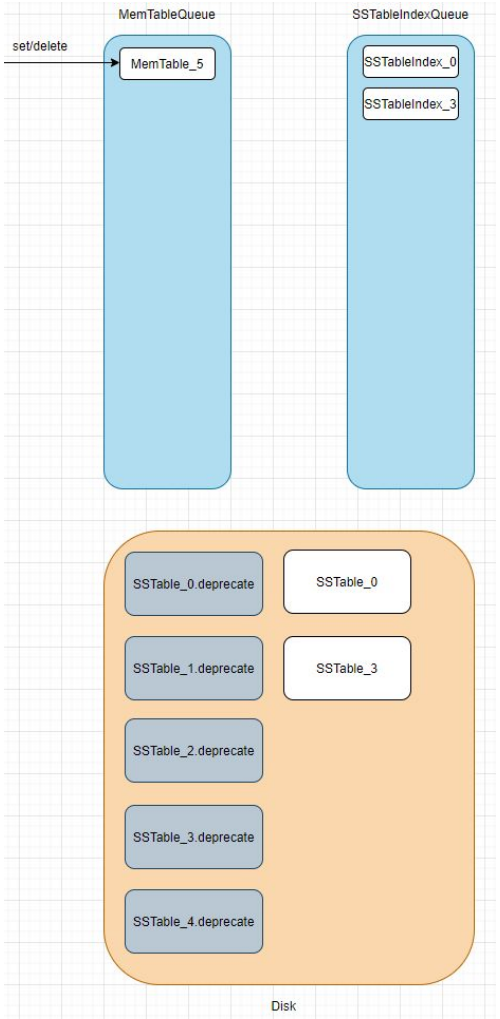

MemTableQueue
SSTableIndexQueue
set/delete
MemTable_0
MemTable_1
SSTableIndex_0
SSTable_0
Disk

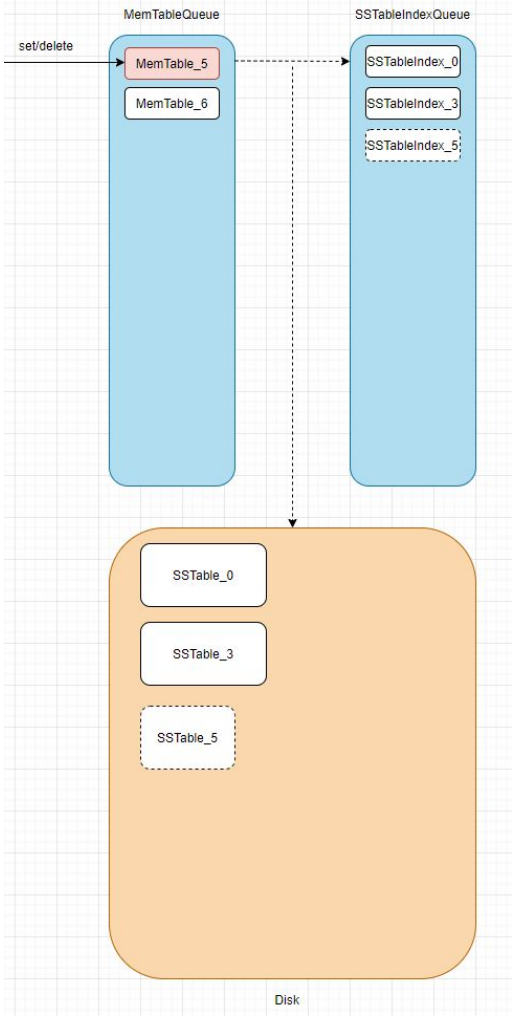

MemTableQueue
SSTableIndexQueue
MemTable_0
SSTableIndex_0
set/delete
MemTable_1
SSTable_0
Disk


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
MemTable_5
SSTableIndexQueue
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_1.deprecate
SSTable_2.deprecate
SSTable_3.deprecate
SSTable_4.deprecate
SSTable_0
SSTable_3
Disk


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

Demo - set and get

```
void test() {
    db_config::MEMTABLE_APPROXIMATE_MAX_SIZE_IN_BYTES = 10;
    db_config::SSTABLE_INDEX_BLOCK_SIZE_IN_BYTES = 8;
    db_config::NUM_SSTABLE_TO_COMPACT = 2;
    db_config::SSTABLE_APPROXIMATE_MAX_SIZE_IN_BYTES = 20;

    ProjectDb db;

    for(auto i = 0; i < 12; i ++) {
        db.set(to_string(i), value: to_string(i) + "-Hello World!");
    }

    for(auto i = 0; i < 12; i ++) {
        const auto key = to_string(i);
        const auto& val = db.get(key);
        if(val.has_value()) {
            cout << val.value() << endl;
        } else {
            cout << "Key: [" << key << "] not found in db!" << endl;
        }
    }
}
```

```
cd cmake-build-release && ./main
0-Hello World!
1-Hello World!
2-Hello World!
3-Hello World!
4-Hello World!
5-Hello World!
6-Hello World!
7-Hello World!
8-Hello World!
9-Hello World!
10-Hello World!
11-Hello World!
```

Demo - set and get

```
[mli@/home/mli/COMSW_4995_Design_Using_C++/COMSW_4995_Design_Using_Cpp_Project/cmake-build-release/projectdb]
[$] ls -lh
total 28K
-rw-r--r-- 1 mli mli 86 Apr  9 12:32 project_db_1.sst
-rw-r--r-- 1 mli mli 53 Apr  9 12:32 project_db_11.sst
-rw-r--r-- 1 mli mli 53 Apr  9 12:32 project_db_12.sst
-rw-r--r-- 1 mli mli  0 Apr  9 12:32 project_db_12.txl
-rw-r--r-- 1 mli mli 86 Apr  9 12:32 project_db_3.sst
-rw-r--r-- 1 mli mli 86 Apr  9 12:32 project_db_5.sst
-rw-r--r-- 1 mli mli 86 Apr  9 12:32 project_db_7.sst
-rw-r--r-- 1 mli mli 86 Apr  9 12:32 project_db_9.sst
```

Demo - set, remove and get

```cpp
void test() {
    db_config::MEMTABLE_APPROXIMATE_MAX_SIZE_IN_BYTES = 10;
    db_config::SSTABLE_INDEX_BLOCK_SIZE_IN_BYTES = 8;
    db_config::NUM_SSTABLE_TO_COMPACT = 2;
    db_config::SSTABLE_APPROXIMATE_MAX_SIZE_IN_BYTES = 20;

    ProjectDb db;

    for(auto i = 0; i < 12; i ++) {
        db.set(to_string(i), value: to_string(i) + "-Hello World!");
    }

    for(auto i = 0; i < 6; i ++) {
        db.remove(to_string(i));
    }

    for(auto i = 0; i < 12; i ++) {
        const auto key = to_string(i);
        const auto& val = db.get(key);
        if(val.has_value()) {
            cout << val.value() << endl;
        } else {
            cout << "Key: [" << key << "] not found in db!" << endl;
        }
    }
}
```

```
cd cmake-build-release && ./main
Key: [0] not found in db!
Key: [1] not found in db!
Key: [2] not found in db!
Key: [3] not found in db!
Key: [4] not found in db!
Key: [5] not found in db!
6-Hello World!
7-Hello World!
8-Hello World!
9-Hello World!
10-Hello World!
11-Hello World!
```

Demo - set, remove and get

```
[mli@/home/mli/COMSW_4995_Design_Using_C++/COMSW_4995_Design_Using_Cpp_Project/cmake-build-release/projectdb]
[$] ls -lh
total 32K
-rw-r--r-- 1 mli mli  86 Apr  9 12:29 project_db_1.sst
-rw-r--r-- 1 mli mli  90 Apr  9 12:29 project_db_11.sst
-rw-r--r-- 1 mli mli 100 Apr  9 12:29 project_db_13.sst
-rw-r--r-- 1 mli mli  58 Apr  9 12:29 project_db_15.sst
-rw-r--r-- 1 mli mli   0 Apr  9 12:29 project_db_15.txl
-rw-r--r-- 1 mli mli  86 Apr  9 12:29 project_db_3.sst
-rw-r--r-- 1 mli mli  86 Apr  9 12:29 project_db_5.sst
-rw-r--r-- 1 mli mli  86 Apr  9 12:29 project_db_7.sst
-rw-r--r-- 1 mli mli  86 Apr  9 12:29 project_db_9.sst
```

Measurements

- Main features of Database:
  - writing - .set()
  - removing - .remove(), similar to .set()
  - reading - .get()
- Supports randomly generated data
- Supports reading data from csv

Data for Benchmark

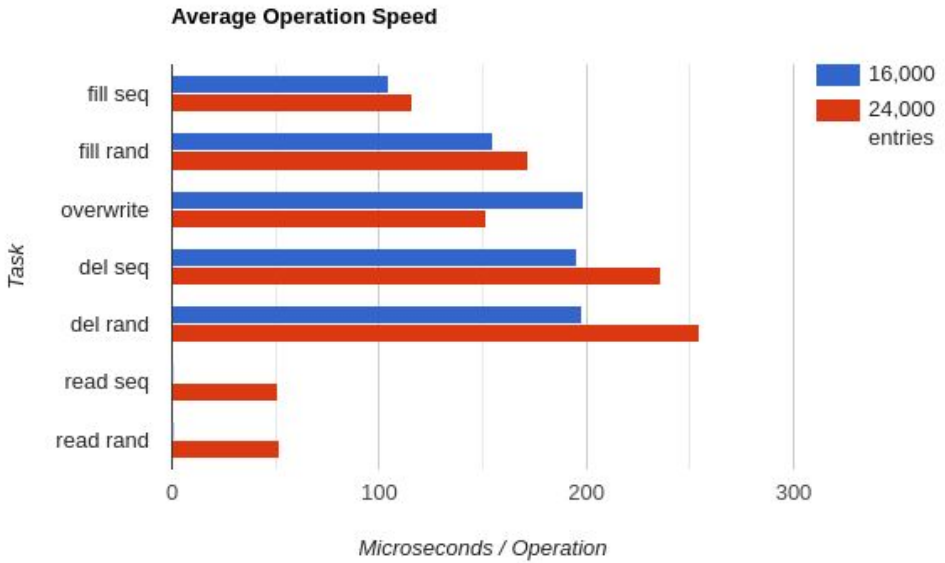
- Key value pairs stored in a vector
- Randomly generated
  - 0, 1, 2, … as keys
  - randomly generated string as values
- Trending YouTube Video Statistics
  - video ID as keys: 2kyS6SvSYSE
  - video title as values: “WE WANT TO TALK ABOUT OUR MARRIAGE”

Benchmark Description

- 7 Tasks:
  - fill sequential / random
  - overwrite
  - delete sequential / random
  - read sequential / random
- Total time for task in microseconds / number of entries

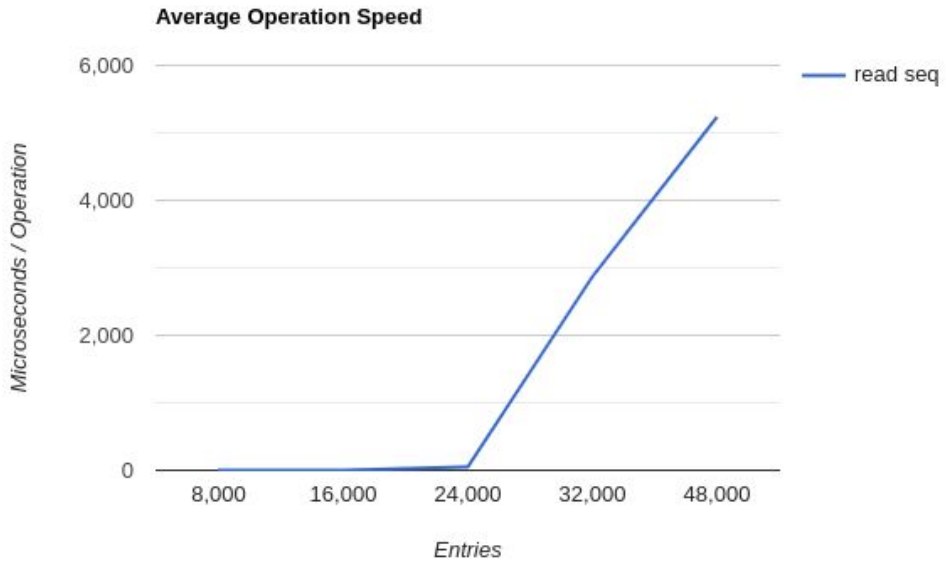
Benchmark: Randomly Generated

- First configuration
  - 2 mb in memory
  - 250 kb segments
- Fill faster than overwrite, delete
- Sequential faster than random
- Slower for more entries
- Overwrite?



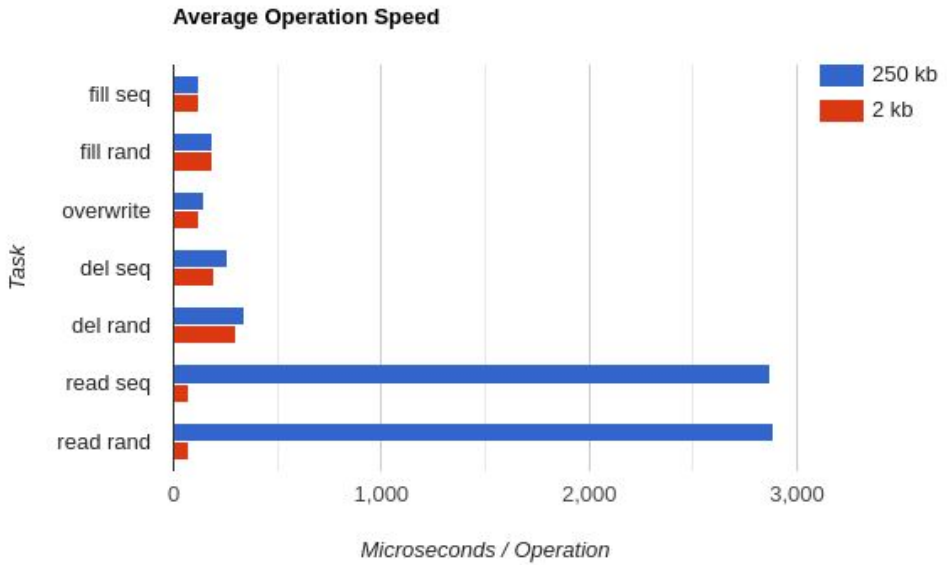
Benchmark: Read

- Fast for low numbers
- Slow when data size > 2 mb
  - values: 100 characters
  - >2.4 mb for 24,000 entries



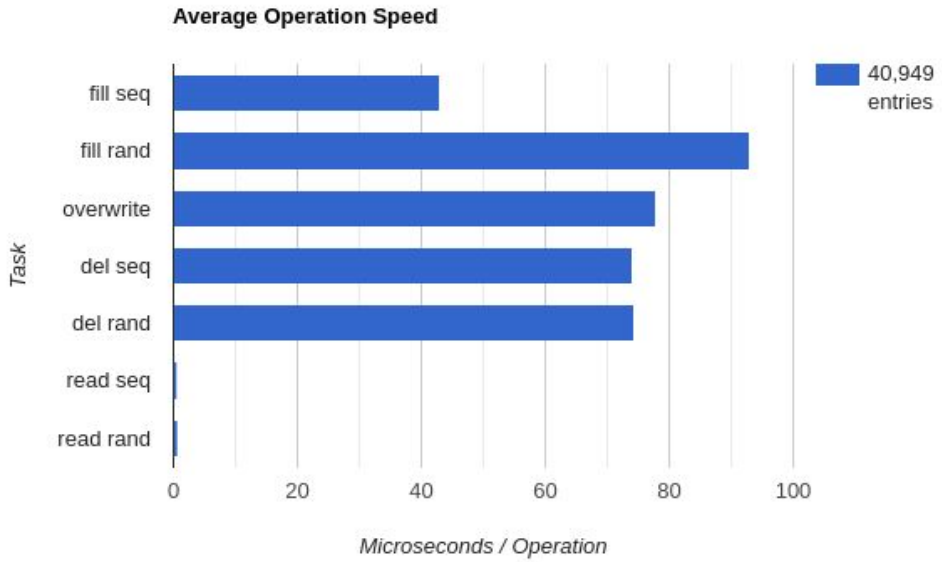
How to Solve This?

- 32,000 entries
- Second configuration
  - 2 mb in memory
  - 2 kb segments
- Much faster read
- Similar performance otherwise
- Very small segments?



Benchmark: Trending YouTube Video Statistics

- Using the first configuration
- Smaller value sizes
- Entirely in memory



Further Benchmarks

- Comparison with LevelDB
- Flush to disk
- More configurations

Future Works

- Multiple potential optimizations
  - Implement Bloom filter to achieve better performance for identifying keys that does not exist
  - Implement optimization for sequential read
    - Block can be kept in memory for a while before throwing away
  - It is possible for us to directly perform the merge while flushing to SSTable, instead of merge SSTable in a separate job
- More benchmarks
- Build on top of the library to actually build a distributed key-value store

Questions?

- LSM-tree
- Workflow
- Implementation
- Measurements
- Future Works