Implementation of LSM-Tree

Yinan An, Minghui Yang, Shun Yao

Achievements

Achievements Overview

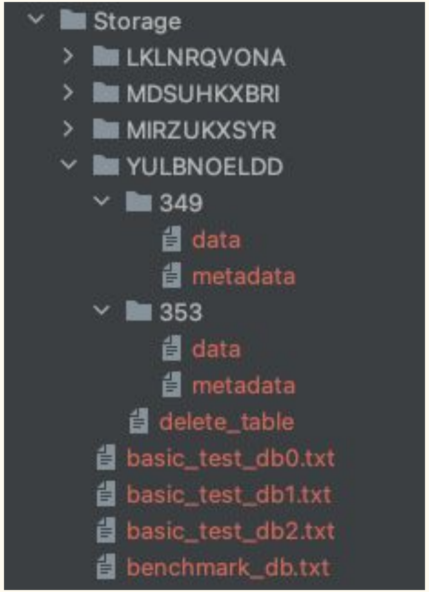
1. LSM-Tree and database basic implementations:
   a. Zone(includes Fence Pointer), Run(includes metadata and data file paths), Level, MemoryTable, DeletedList(for RangeDelete)
   b. Leveling and Tiering compaction strategies
   c. Single Query, Range Query, Single Delete, Range Delete, Put
   d. Extend basic_test: add tests for “RangeDelete”
   e. Durable database and compatibility with multiple databases
2. SST data file saved in binary format and read files in blocks
3. Compatibility with multiple databases
4. Perform basic functionality tests, durable tests and different experiments

Highlight
&
Implementation

Database initialization and multiple databases

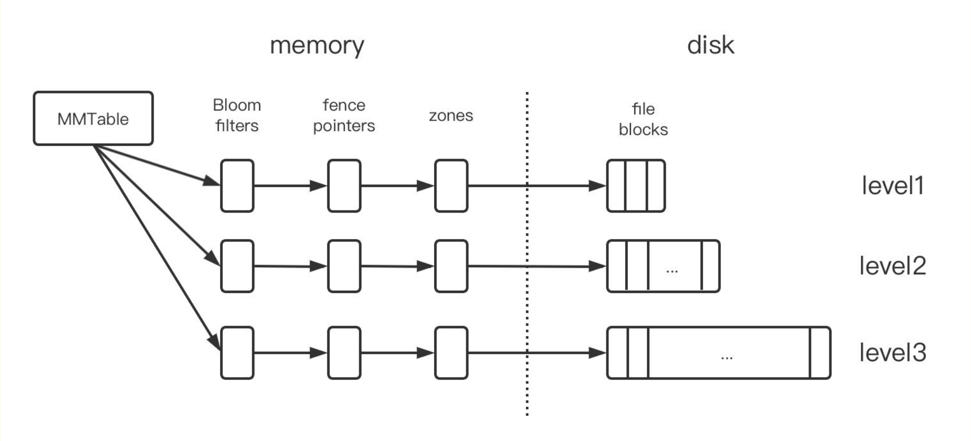
When opening a database

- Read the config file with its name the same as database name. If the config file doesn't exist, create a new data directory for this database and a config file with default settings.
- Construct Levels of the database
- Load metadata of each Run into the memory. And load delete list.



```
Storage
  LKLNRQVONA
  MDSUHKXBRI
  MIRZUKXSYR
  YULBNOELDD
    349
      data
      metadata
    353
      data
      metadata
    delete_table
  basic_test_db0.txt
  basic_test_db1.txt
  basic_test_db2.txt
  benchmark_db.txt
```

Zone and Binary file



Zone is to record the min/max key and the byte range(min/max byte) of every data block(file block).

Create zone from data map:

number of elements per zone: N

Every key-value pair in map has a byte offset relative to the start of map

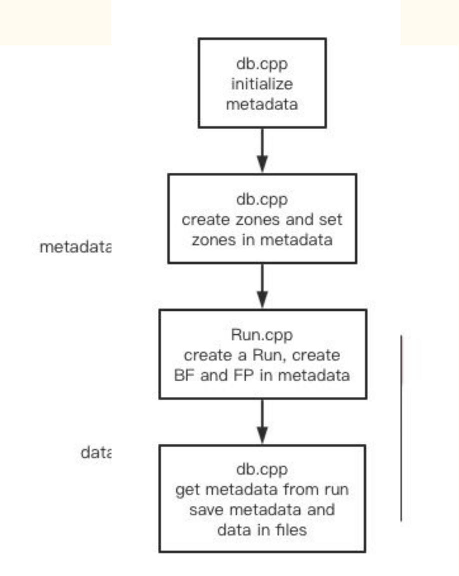
For every N elements in map, create a zone to record min/max key and min/max byte based on offset.

Write/Read file in binary format: std::ios::binary

Read file block based on offset:

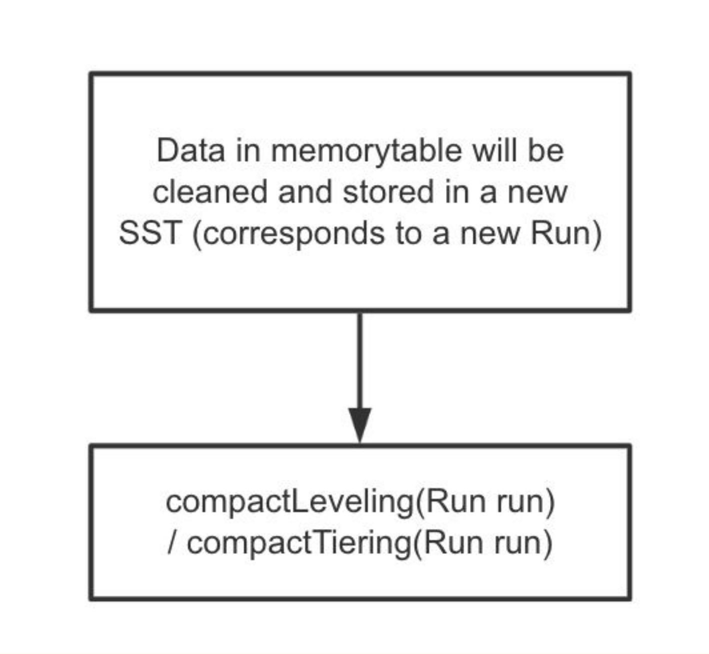
seekg(offset, std::ios::beg)

To Save a Run in disk:



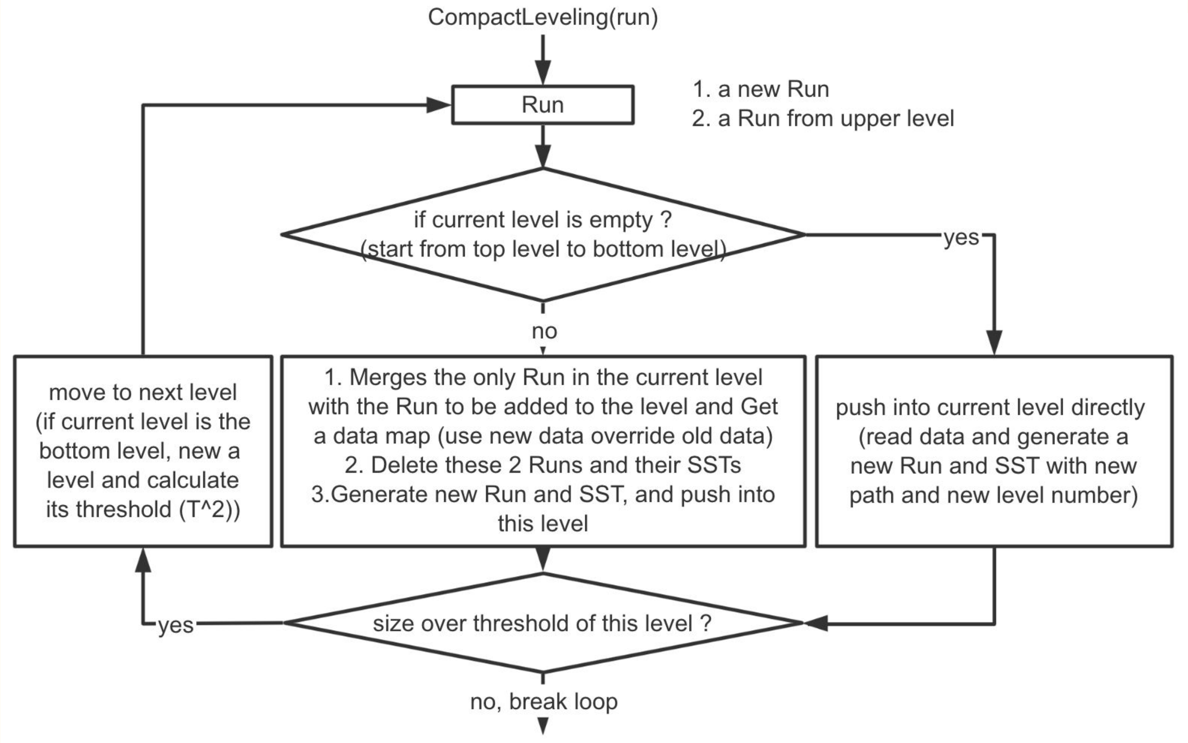
Leveling and Tiering Compaction

- **Timing**: put() (check threshold of memorytable) & close()


Data in memorytable will be cleaned and stored in a new SST (corresponds to a new Run)
compactLeveling(Run run) / compactTiering(Run run)

Leveling and Tiering Compaction

- **Leveling Strategy:**



CompactLeveling(run)

Run

1. a new Run
2. a Run from upper level

if current level is empty ?
(start from top level to bottom level)

yes

push into current level directly (read data and generate a new Run and SST with new path and new level number)

no

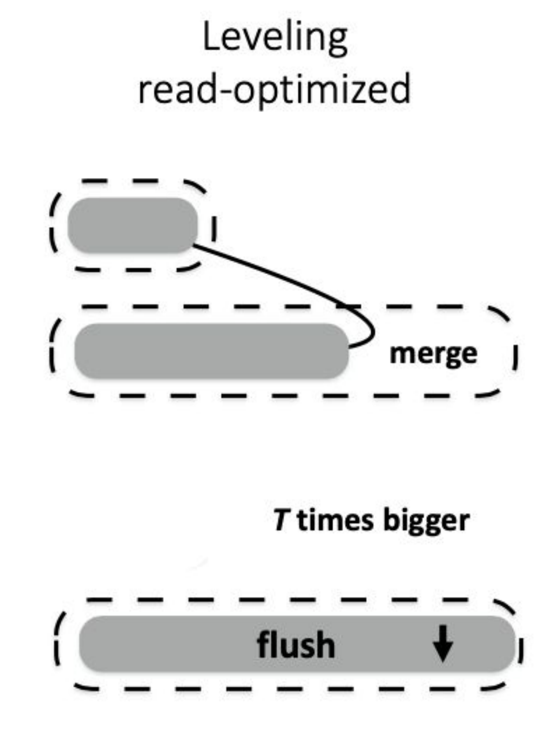
1. Merges the only Run in the current level with the Run to be added to the level and Get a data map (use new data override old data)
2. Delete these 2 Runs and their SSTs
3.Generate new Run and SST, and push into this level

size over threshold of this level ?

yes

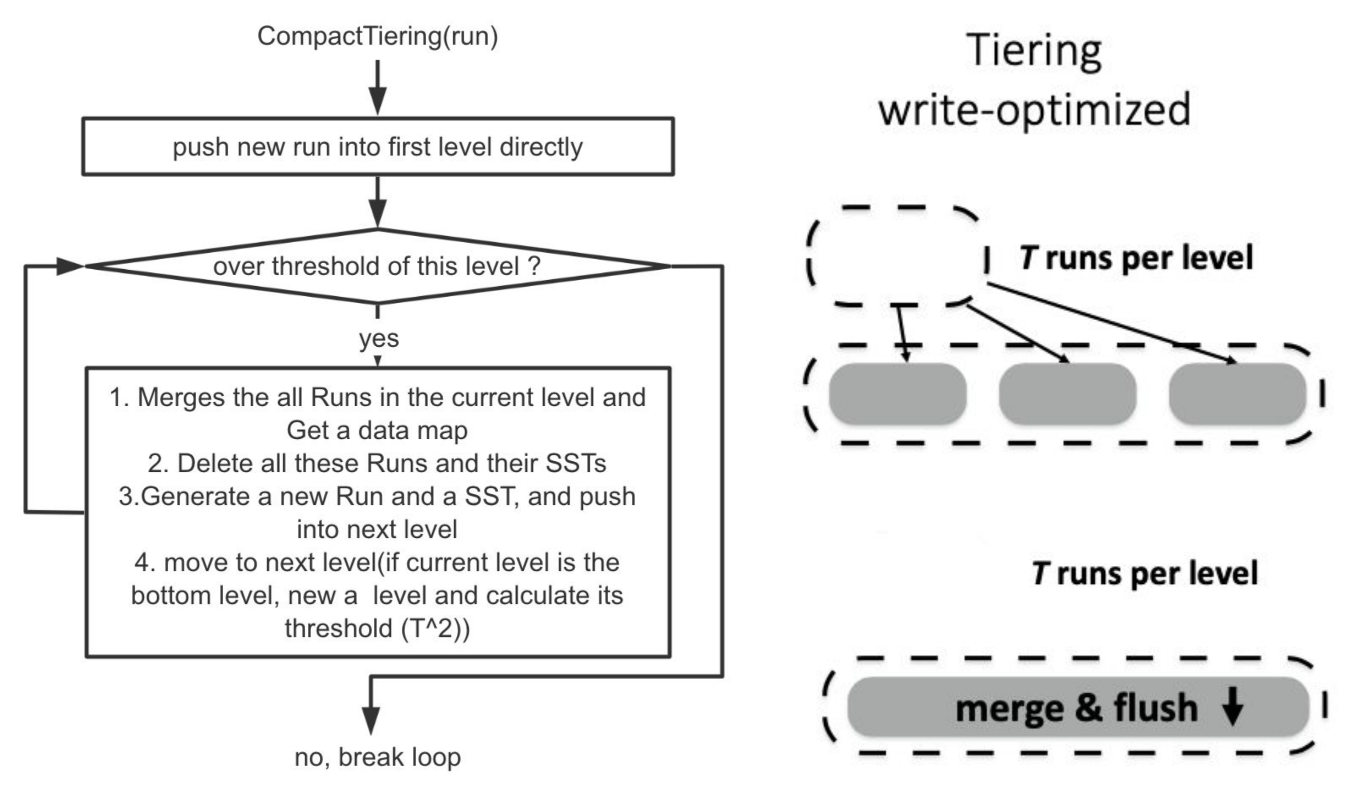
move to next level (if current level is the bottom level, new a level and calculate its threshold (T^2))

no, break loop



Leveling and Tiering Compaction

- **Tiering Strategy:**



CompactTiering(run)

push new run into first level directly

over threshold of this level ?

yes

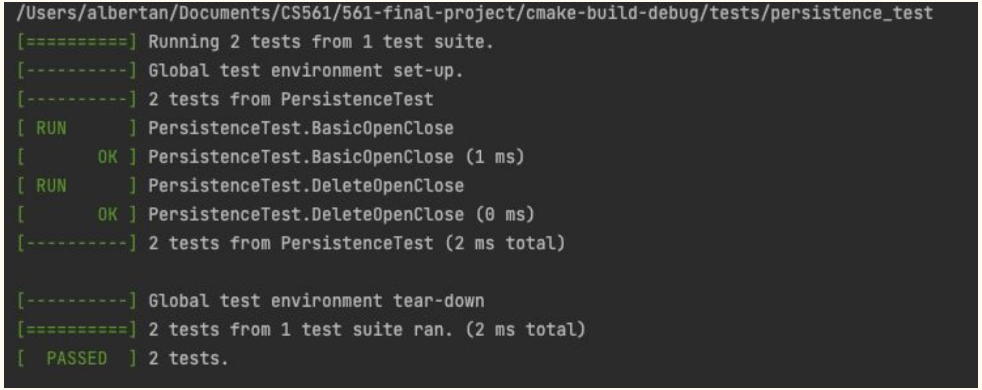
1. Merges the all Runs in the current level and Get a data map
2. Delete all these Runs and their SSTs
3.Generate a new Run and a SST, and push into next level
4. move to next level(if current level is the bottom level, new a level and calculate its threshold (T^2))

no, break loop

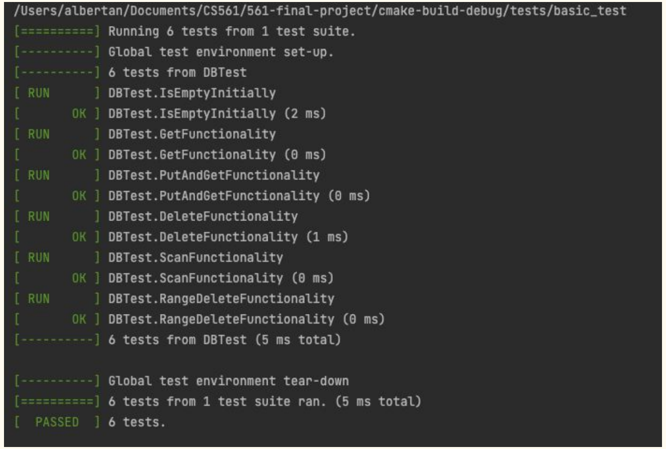
Experiments

Basic Test, Persistent Test and Durable Test

All following tests and experiments are performed under
CPU: M1-pro   OS: MacOS 12.0   Cpp17

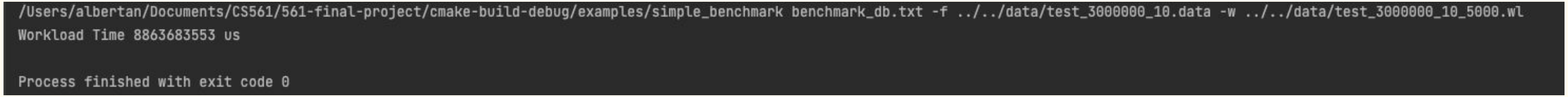


```
/Users/albertan/Documents/CS561/561-final-project/cmake-build-debug/tests/persistence_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from PersistenceTest
[ RUN      ] PersistenceTest.BasicOpenClose
[       OK ] PersistenceTest.BasicOpenClose (1 ms)
[ RUN      ] PersistenceTest.DeleteOpenClose
[       OK ] PersistenceTest.DeleteOpenClose (0 ms)
[----------] 2 tests from PersistenceTest (2 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (2 ms total)
[  PASSED  ] 2 tests.
```



```
/Users/albertan/Documents/CS561/561-final-project/cmake-build-debug/tests/basic_test
[==========] Running 6 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 6 tests from DBTest
[ RUN      ] DBTest.IsEmptyInitially
[       OK ] DBTest.IsEmptyInitially (2 ms)
[ RUN      ] DBTest.GetFunctionality
[       OK ] DBTest.GetFunctionality (0 ms)
[ RUN      ] DBTest.PutAndGetFunctionality
[       OK ] DBTest.PutAndGetFunctionality (0 ms)
[ RUN      ] DBTest.DeleteFunctionality
[       OK ] DBTest.DeleteFunctionality (1 ms)
[ RUN      ] DBTest.ScanFunctionality
[       OK ] DBTest.ScanFunctionality (0 ms)
[ RUN      ] DBTest.RangeDeleteFunctionality
[       OK ] DBTest.RangeDeleteFunctionality (0 ms)
[----------] 6 tests from DBTest (5 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 1 test suite ran. (5 ms total)
[  PASSED  ] 6 tests.
```

Leveling, Number of elements per zone: 50, First level threshold: 50, MMTable: 50



```
/Users/albertan/Documents/CS561/561-final-project/cmake-build-debug/examples/simple_benchmark benchmark_db.txt -f ../../data/test_3000000_10.data -w ../../data/test_3000000_10_5000.wl
Workload Time 8863683553 us

Process finished with exit code 0
```

Finish 3 million insertions and 3 million operations with data-dimension equals to 10 in 8864s (about 2.5h)

Exp-1: Evaluate Base Threshold of Levels

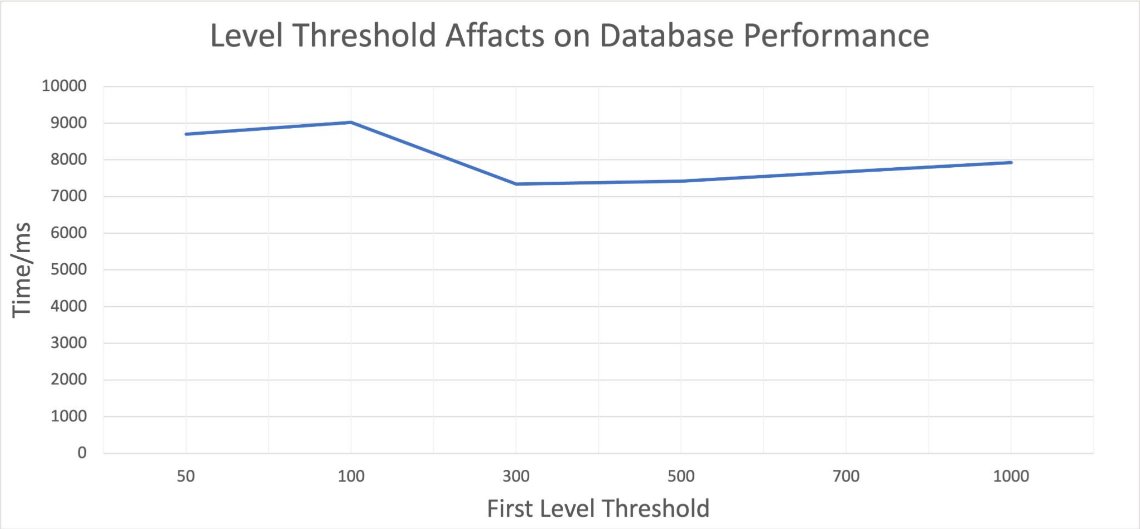
Leveling strategy

Number of elements per zone: 50

MMTable size: 25

test_10000_3.data

test_10000_3_2000.wl



Exp-2: Evaluate No. of Elements per Zone

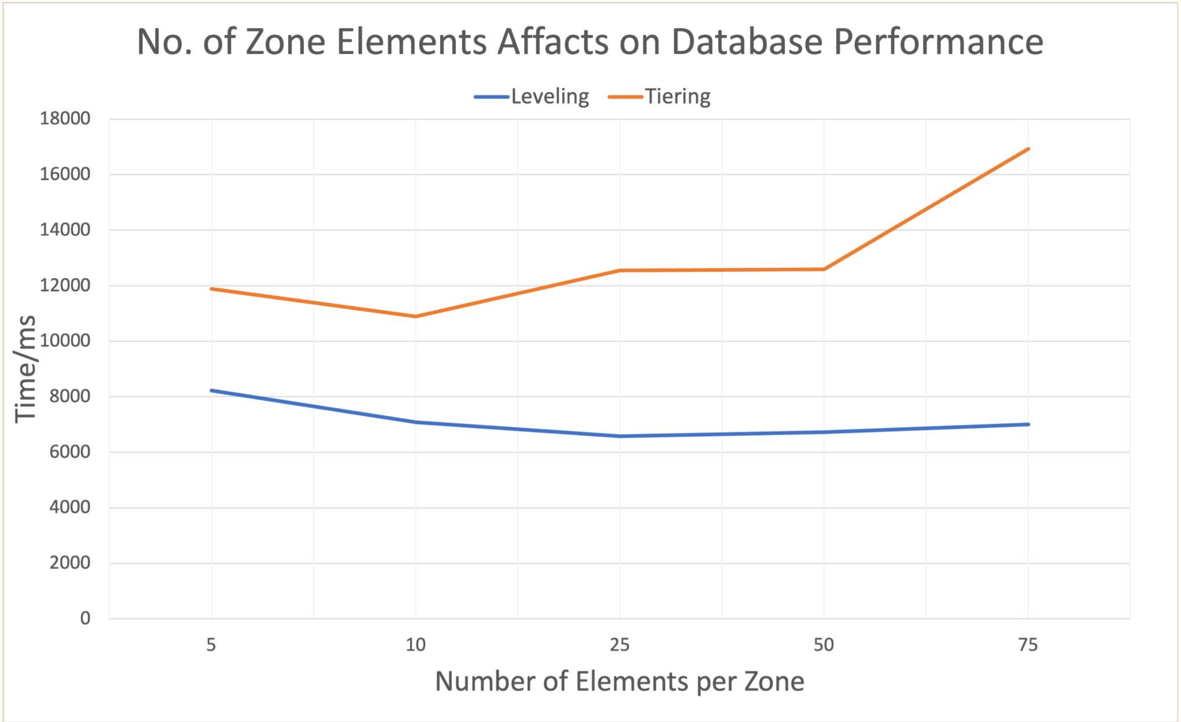
Leveling and Tiering strategies

First level threshold: 25

MMTable size: 25

test_10000_3.data

test_10000_3_2000.wl



Exp-3: Evaluate BF size

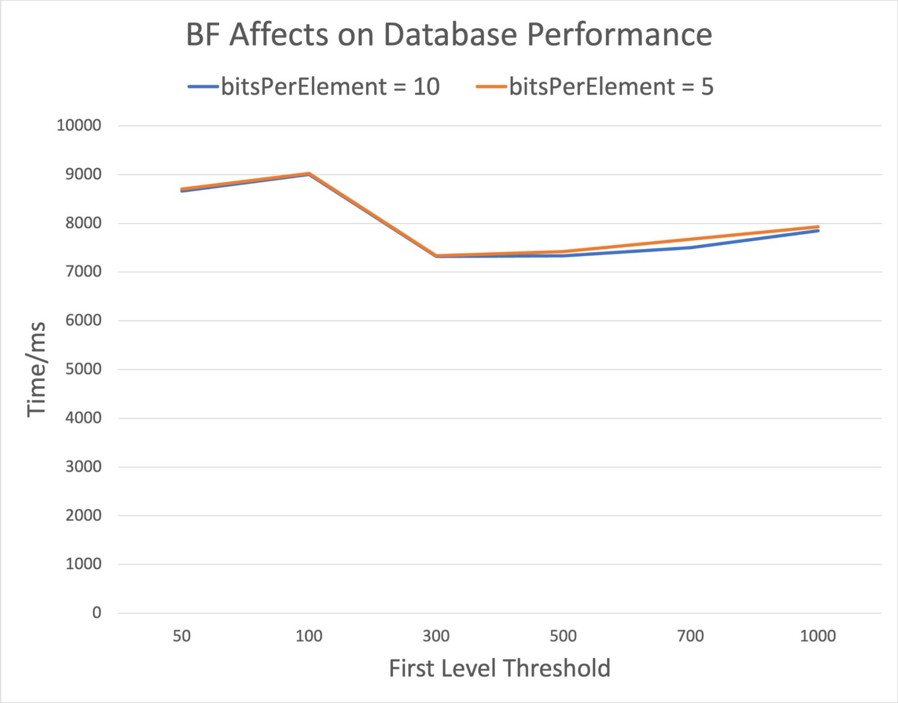
Leveling strategy

Number of elements per zone: 50

MMTable size: 25

test_10000_3.data

test_10000_3_2000.wl



Exp-4 Evaluate Key Distribution

Leveling strategy

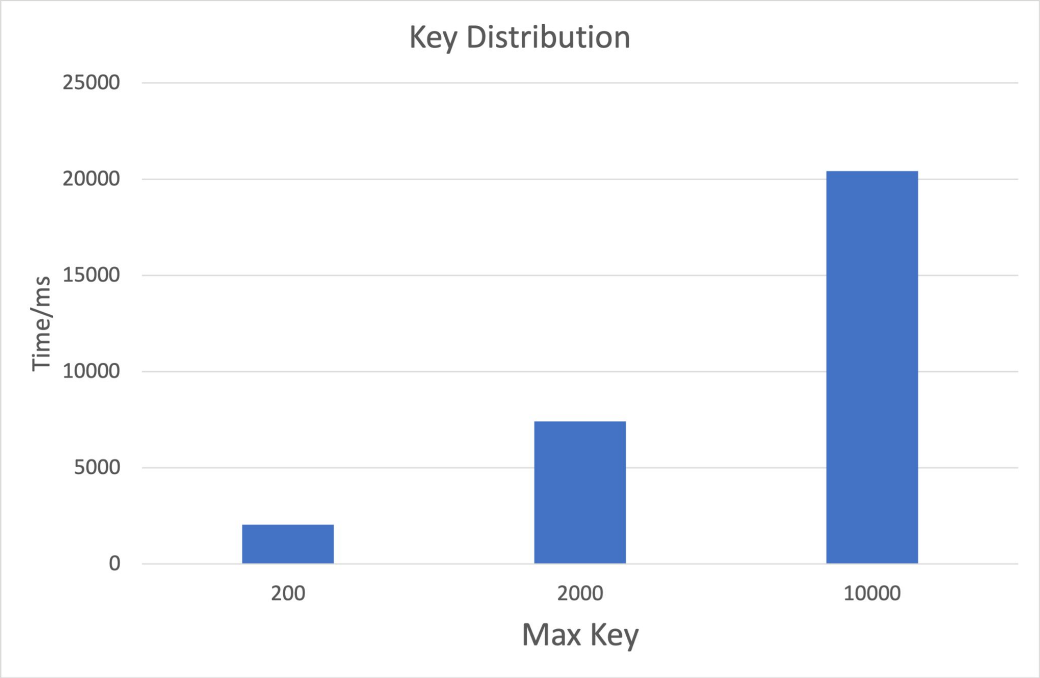
Number of elements per zone: 50

First level threshold: 50

MMTable size: 25

test_10000_3.data

test_10000_3_200.wl
test_10000_3_2000.wl
test_10000_3_10000.wl



Exp-5: Reading/Writing Cost

Leveling strategy

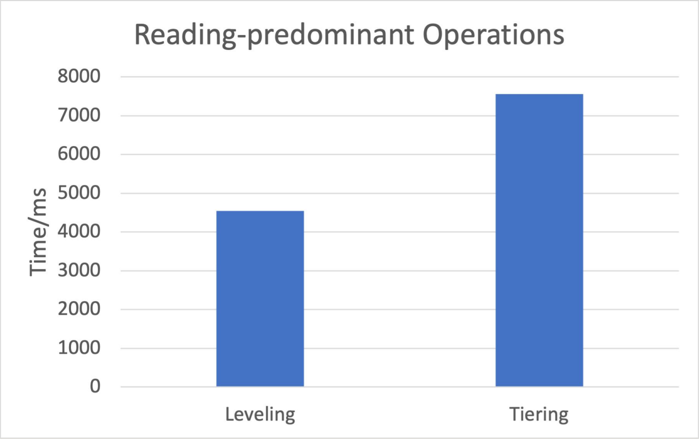
Number of elements per zone: 50

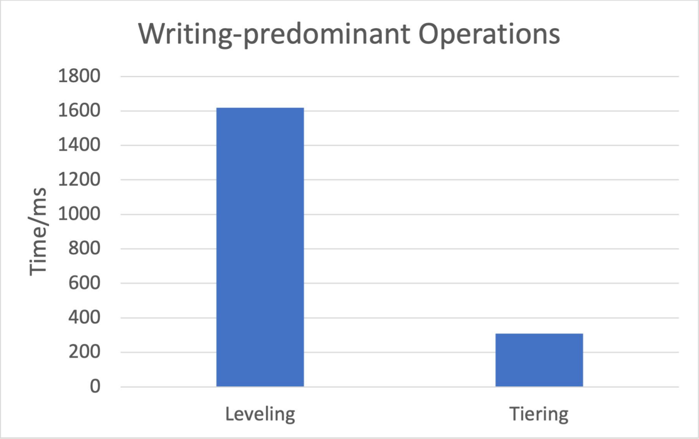
First level threshold: 50

MMTable size: 25

Reading-predominant:
test_100_3.data   test_10000_3_2000.wl

Writing-predominant:
test_10000_3.data test_100_3_200.wl





Experience & Challenges

1. **Mechanism of Range Delete**: Due to the lack of information, we spent a long time to figure out that we need to add timestamps in our database and each Value, contain a list of deleted records (start, end, timestamp) and modify the query methods.
2. **Compact Leveling and Tiering**: Although we understand the fundamental concepts of both approaches, there are a lot of details to deal with when writing code.
3. **Google test and CMake**: We had no experience with CMake and Google Test. So, it took us a long time to modify the CMakeLists.txt and find out the workflow of Google test. For example, when we were trying to run "basic_test", value in ASSERT_EQ and EXPECT_EQ are always different from what we expect, until we find out that the methods Setup() ran before every test.
4. **Data store in binary format:** Reading string and vector from binary files is challenging at first because the size of it is uncertain. So when we write string and vector to file, we first write the size, it will help us to read later. Also we have no experience in reading from a file block instead of the entire file, then we find the function seekg is useful to read file from a specific offset.

Thank You!