

SSTable

Jongki Park, Sanghyun Cho, Jayoung Cho

E-Mail: jkipark@dankook.ac.kr

98shcho@naver.com

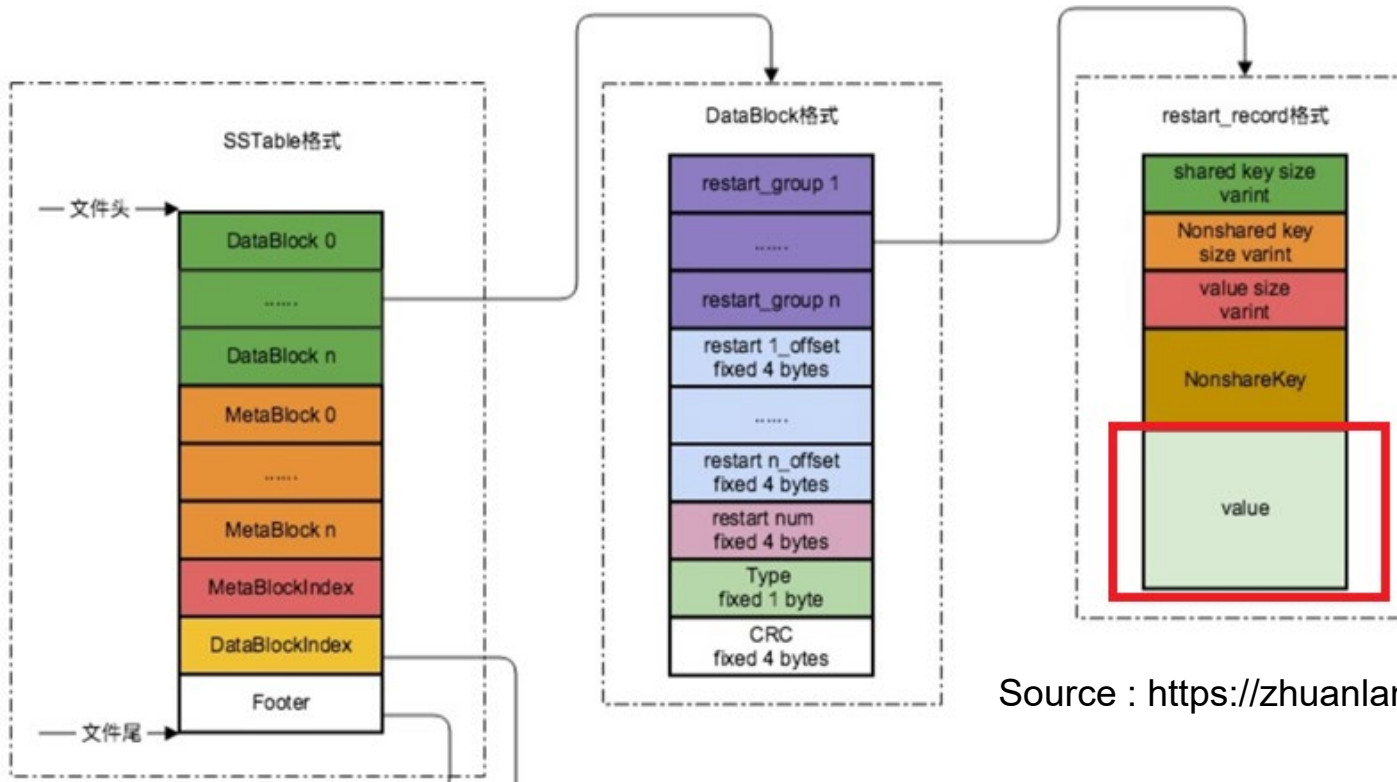
26006@naver.com

1. Entry size in SSTable
2. Filter block in SSTable
3. Further research
4. Appendix

- SSTable
= A fixed size “file” with well defined format
(default 2MB)
- Focused on getting familiar with SSTable’s format and SSTable related options

1. Entry size in SSTable

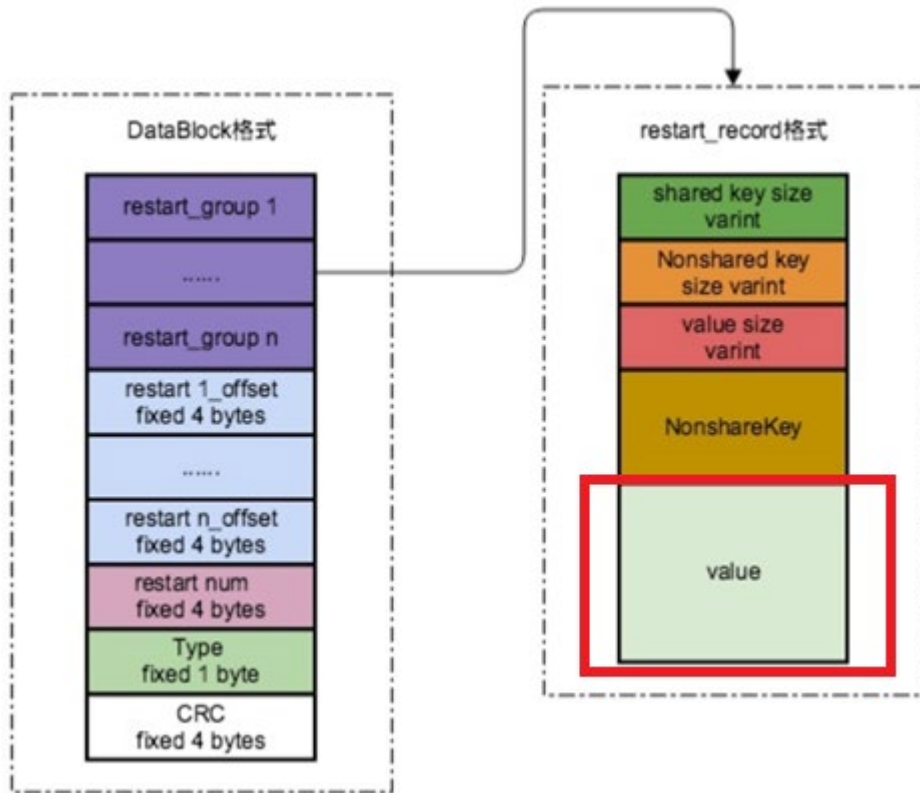
- How does changing the key size affect the Level DB 's write and read performance?



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

Entry size in SSTable

- Fillrandom performance - Expectation



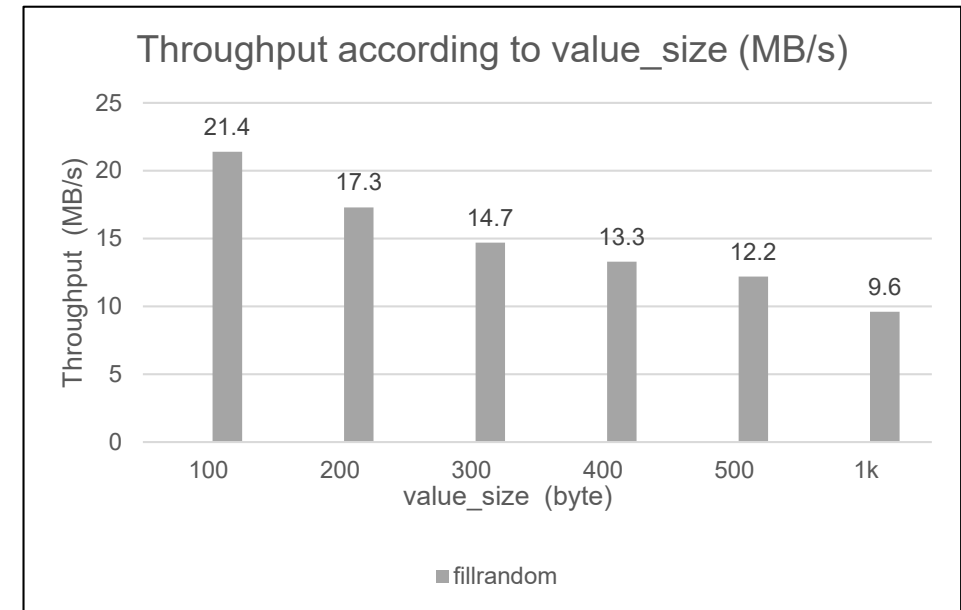
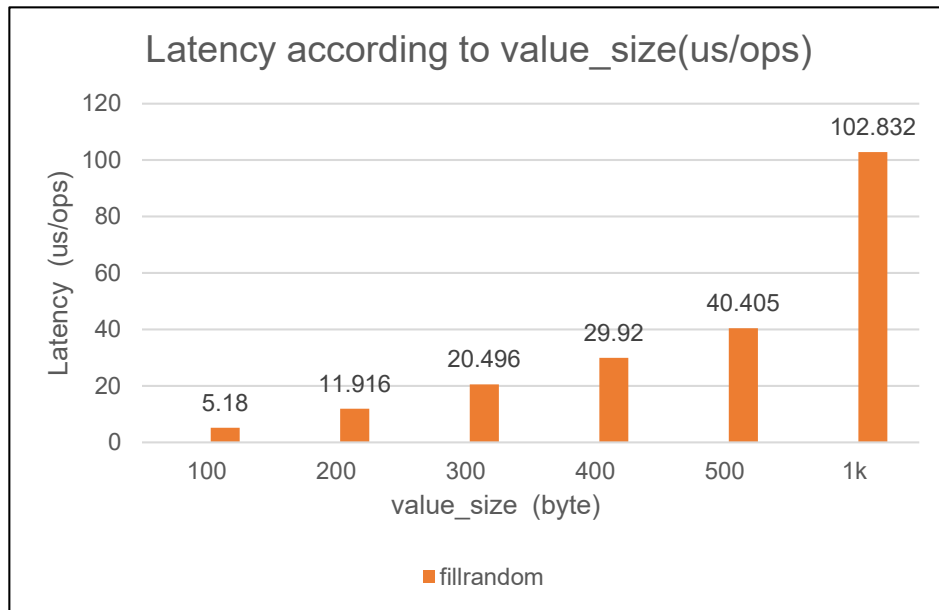
If the size of value increases,
the number of entries that can fit in one file decreases
when storing the same number of entries



More SSTables are needed

Entry size in SSTable

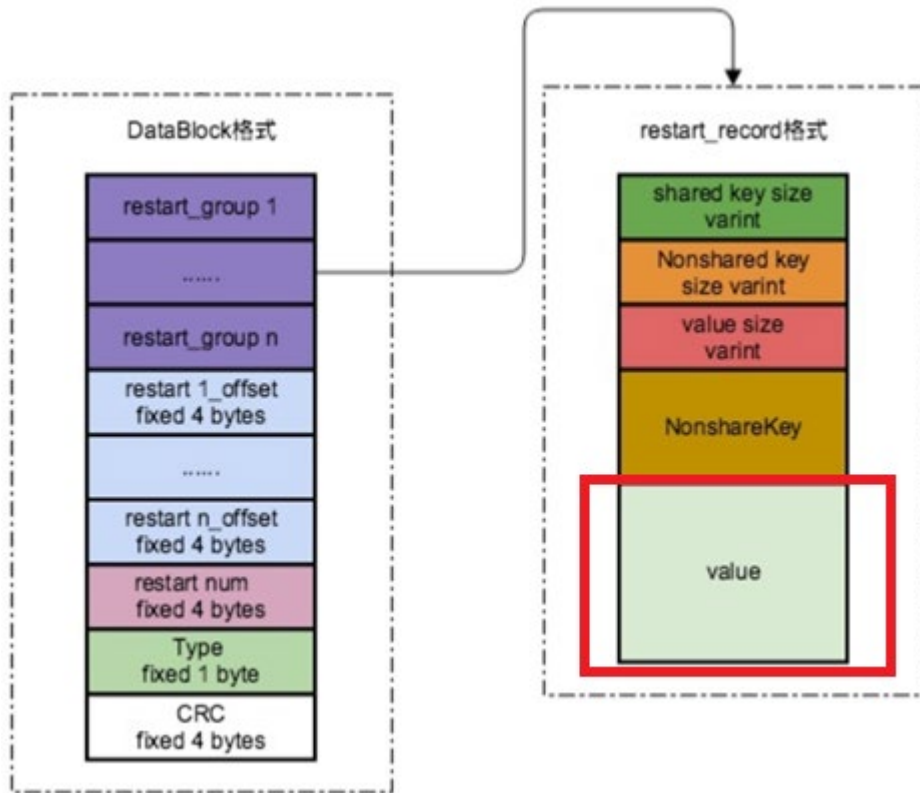
- Fillrandom performance - Consulsion



Number of <Key, Value> entries = 2,000,000

Entry size in SSTable

- Readrandom performance - Expectation



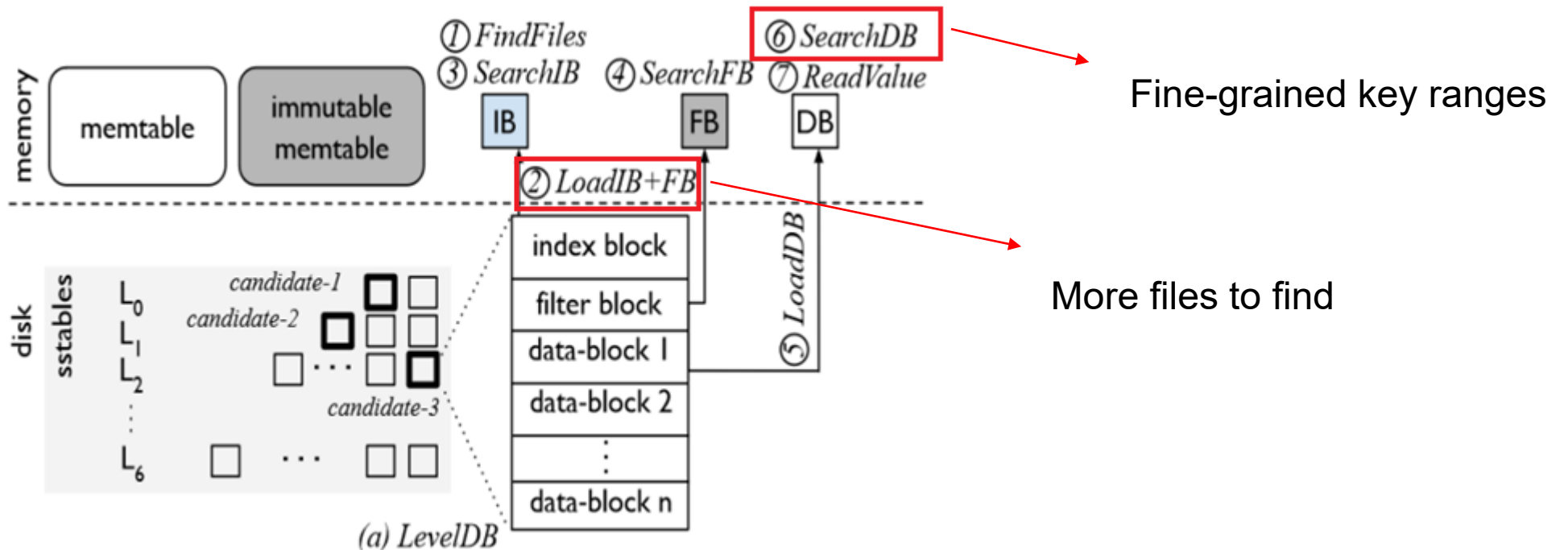
If the size of value increases, more SSTables are needed



More files to find vs Fine-grained key ranges

Entry size in SSTable

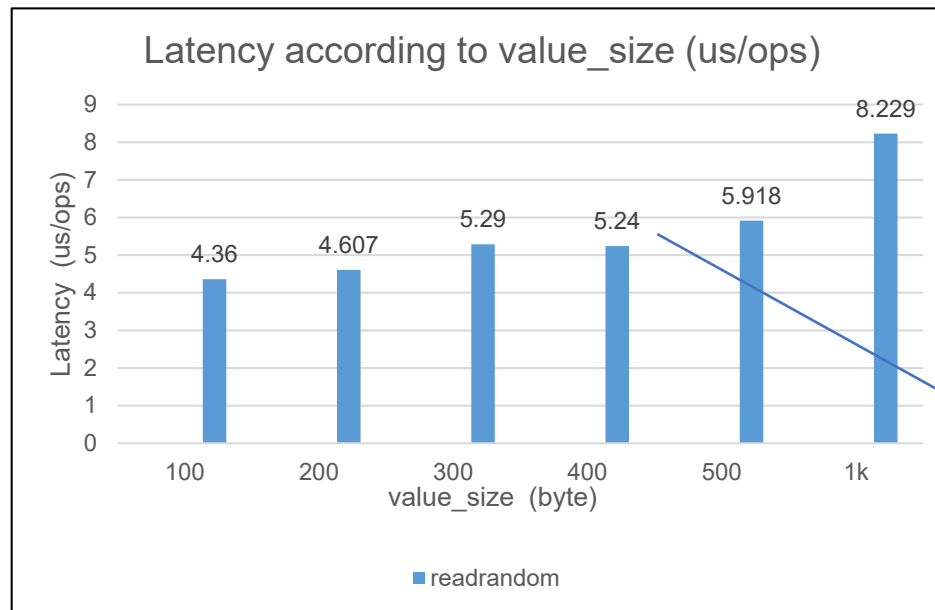
- More files to find vs Fine-grained key ranges
 - -> It depends on how to find the key in Level DB



Source : Yifan Dai, From WiscKey to Bourbon, OSDI '20

Entry size in SSTable

- Readrandom performance – Conclusion
 - More effected by the number of files (Load)

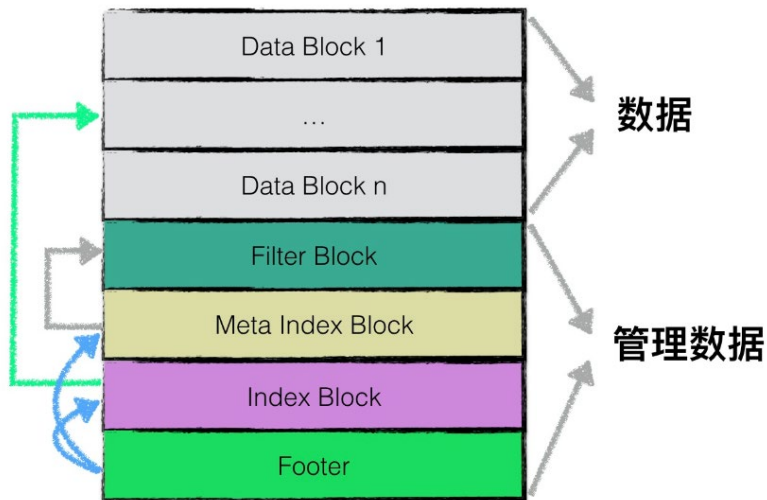


Maybe because of the random read

Number of <Key, Value> entries = 2,000,000

2. Filter block in SSTable

- How does applying bloom filter affect write performance of Level DB?



논리적 구조

논리적으로 다른 기능에 따라 leveldb는 sstable을 논리적으로 다음과 같이 나눕니다.

1. 데이터 블록 : 키 값 데이터 쌍을 저장하는 데 사용됩니다.
2. 필터 블록 : 일부 필터 관련 데이터(블룸 필터)를 저장하는 데 사용되지만 사용자가 필터를 사용하도록 leveldb를 지정하지 않으면 leveldb는 이 블록에 내용을 저장하지 않습니다.
3. 메타 인덱스 블록 : 필터 블록의 인덱스 정보를 저장하는 데 사용됩니다(인덱스 정보는 sstable 파일의 오프셋 및 데이터 길이를 나타냄).
4. 인덱스 블록 : 각 데이터 블록의 인덱스 정보를 저장하는 데 사용되는 인덱스 블록.
5. 바닥글 : 메타 인덱스 블록과 인덱스 블록의 인덱스 정보를 저장하는 데 사용됩니다.

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

Filter block in SSTable

■ Verification - ufttrace

```
# Function Call Graph for 'db_bench' (session: eaadae18b97901f0)
===== FUNCTION CALL GRAPH =====
# TOTAL TIME  FUNCTION
26.350 ms : (1) db_bench
26.350 ms : (832) leveldb::TableBuilder::Add
603.007 us : +- (832) leveldb::TableBuilder::ok
211.437 us : | +- (832) leveldb::TableBuilder::status
41.668 us : | | (832) leveldb::Status::Status
: |
43.086 us : | +- (832) leveldb::Status::ok
: |
44.460 us : | +- (832) leveldb::Status::~Status
: |
42.529 us : +- (832) leveldb::Slice::size
: |
42.658 us : +- (832) leveldb::Slice::data
: |
78.898 us : +- (832) std::__cxx11::basic_string::assign
: |
14.534 ms : +- (854) leveldb::BlockBuilder::Add
763.115 us : | +- (1708) leveldb::Slice::Slice
112.917 us : | | +- (1708) std::__cxx11::basic_string::data
: | |
103.981 us : | | +- (1708) std::__cxx11::basic_string::size
: | |
55.146 us : | +- (854) std::__cxx11::basic_string::empty
: |
206.426 us : +- (4138) leveldb::Slice::size
: |
39.810 us : +- (788) std::min
: |
```

<No apply Bloom Filter>

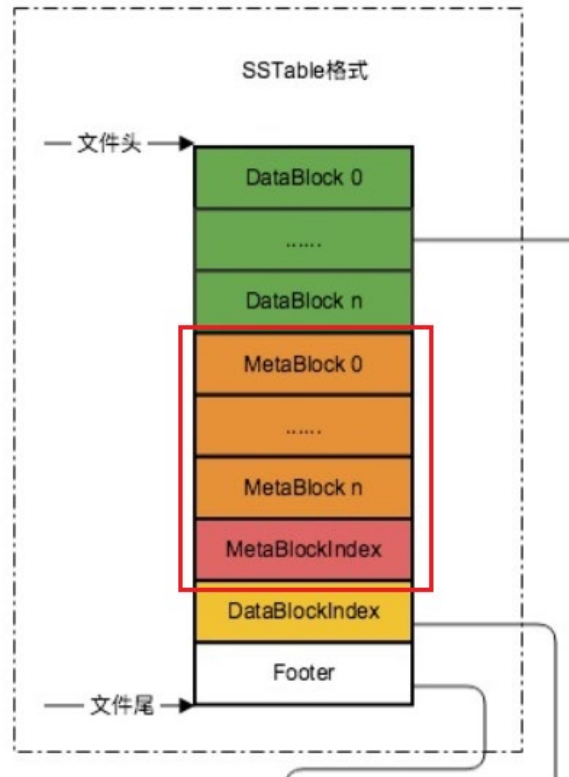
```
# Function Call Graph for 'db_bench' (session: 348e5301f06231d3)
===== FUNCTION CALL GRAPH =====
# TOTAL TIME  FUNCTION
32.790 ms : (1) db_bench
32.790 ms : (832) leveldb::TableBuilder::Add
631.630 us : +- (832) leveldb::TableBuilder::ok
242.891 us : | +- (832) leveldb::TableBuilder::status
41.870 us : | | (832) leveldb::Status::Status
: |
43.262 us : | +- (832) leveldb::Status::ok
: |
43.922 us : | +- (832) leveldb::Status::~Status
: |
1.919 ms : +- (832) leveldb::FilterBlockBuilder::AddKey
51.254 us : | +- (832) std::__cxx11::basic_string::size
: |
1.217 ms : | +- (832) std::vector::push_back
43.229 us : | | +- (832) std::move
: | |
961.565 us : | | +- (832) std::vector::emplace_back
41.798 us : | | | +- (832) std::forward
: | | |
3.128 us : | | | +- (14) std::vector::end
0.679 us : | | | | (14) __gnu_cxx::__normal_iterator::__normal_iterator
: | | |
133.024 us : | | +- (14) std::vector::_M_realloc_insert
37.862 us : | | | +- (14) std::vector::_M_check_len
24.667 us : | | | | +- (28) std::vector::max_size
1.471 us : | | | | | +- (28) std::_Vector_base::_M_get_Tp_allocator
: | | |
```

<Apply Bloom Filter>

See TableBuilder::Add() in ~/leveldb/table/table_builder.cc

Filter block in SSTable

- Fillseq performance – Expectation



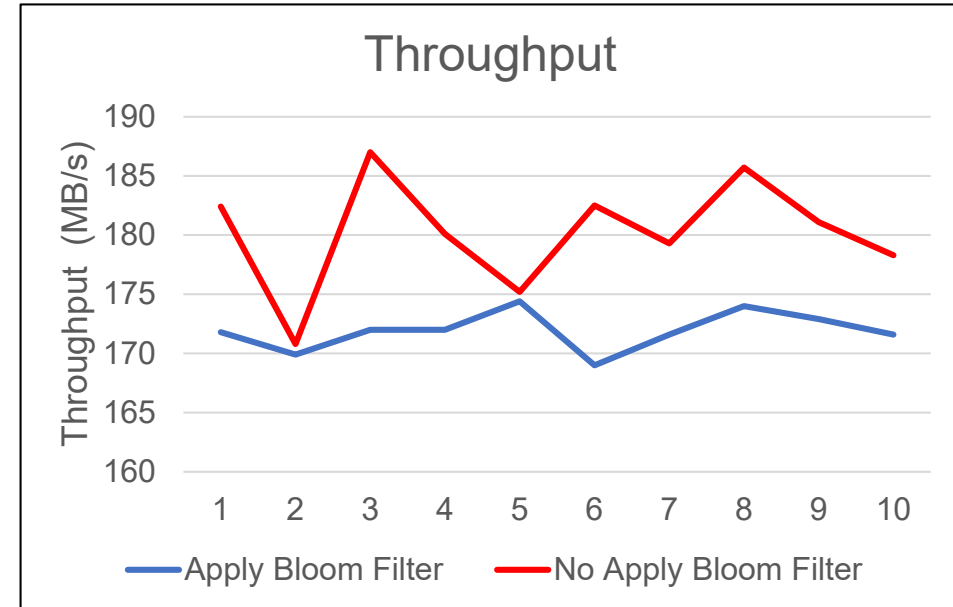
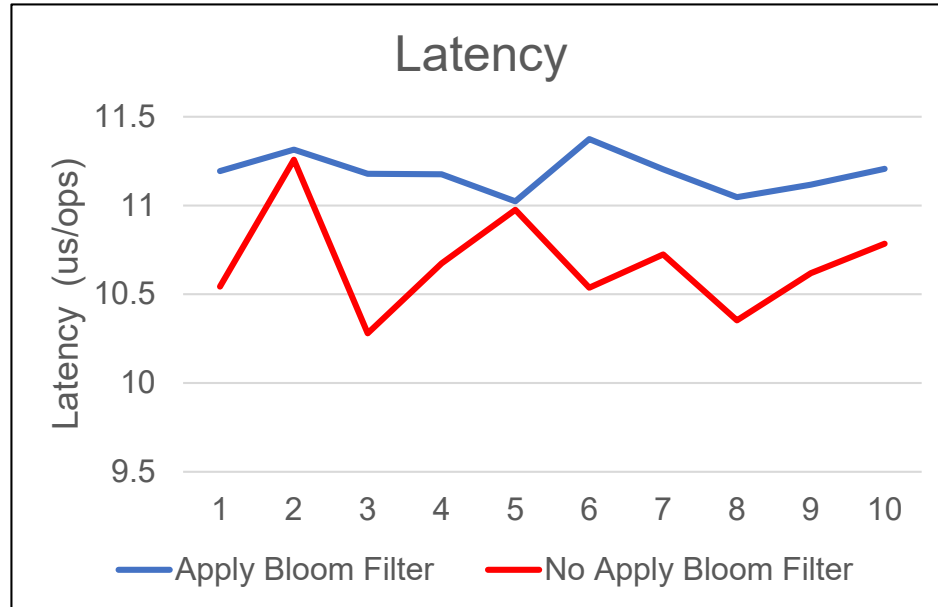
If filter block is not applied,
there is no need to write additional data in meta block



Latency and Throughput will be improved when writing

Filter block in SSTable

- Fillseq performance – Conclusion



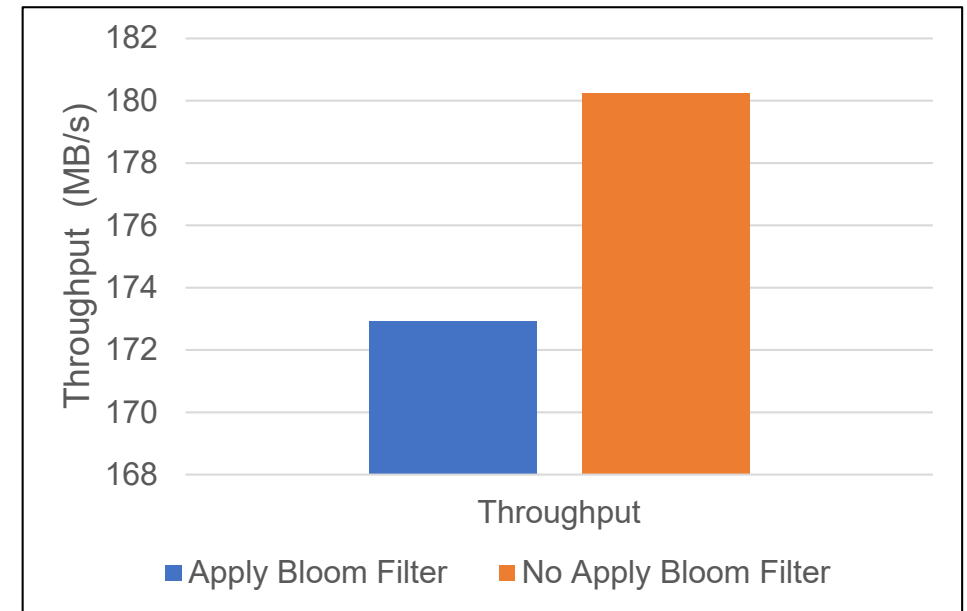
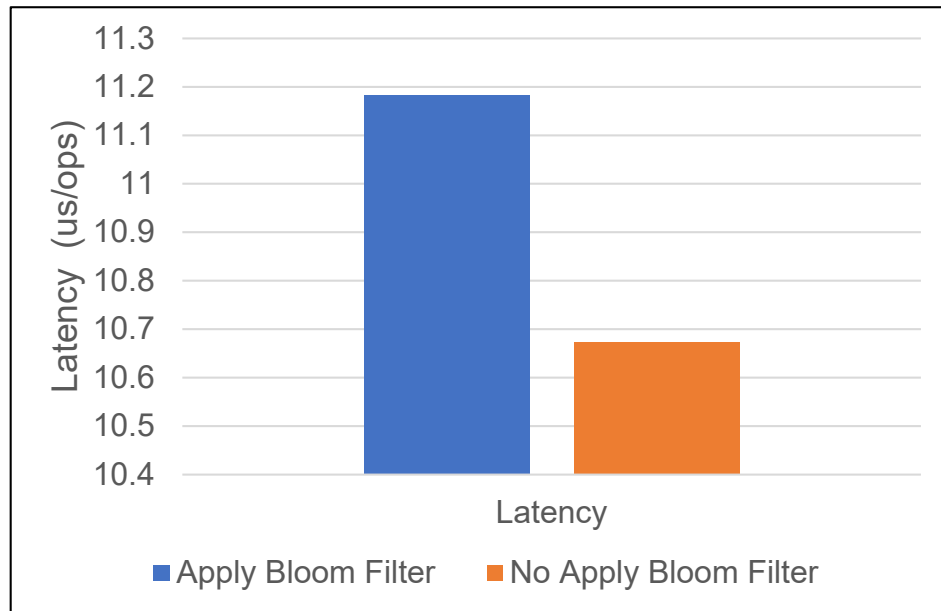
Value_size = 2000

Filter block in SSTable

- Fillseq performance – Conclusion

Apply Bloom Filter → average latency is 10.674 micros/op, average throughput is 180.24MB/s

No Apply Bloom Filter → average latency is 11.183 micros/op, average throughput is 172.92MB/s



Filter block in SSTable

■ Discussion

- Result : Latency → 약 0.5micros 감소, Throughput → 약 7MB 증가
- We expected a dramatic change, but it didn't show much change. Why?

✓ 1. Output

- The output of db_bench doesn't mean "time taken to create a SSTable"
- It just means "time taken to write a single key-value pair"

✓ 2. Fillseq

- Sequential key order → No compaction → Reduces the amount of SSTable created

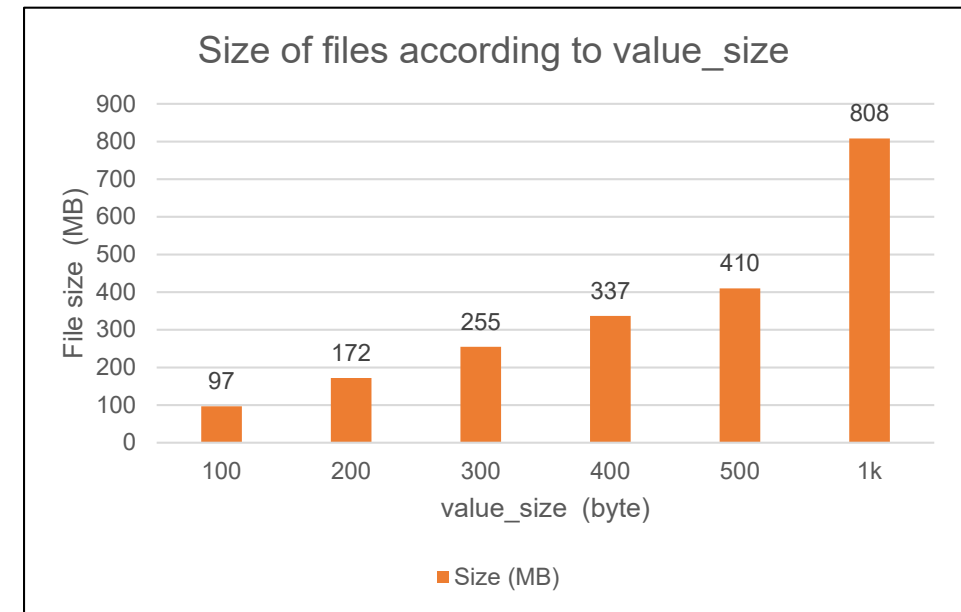
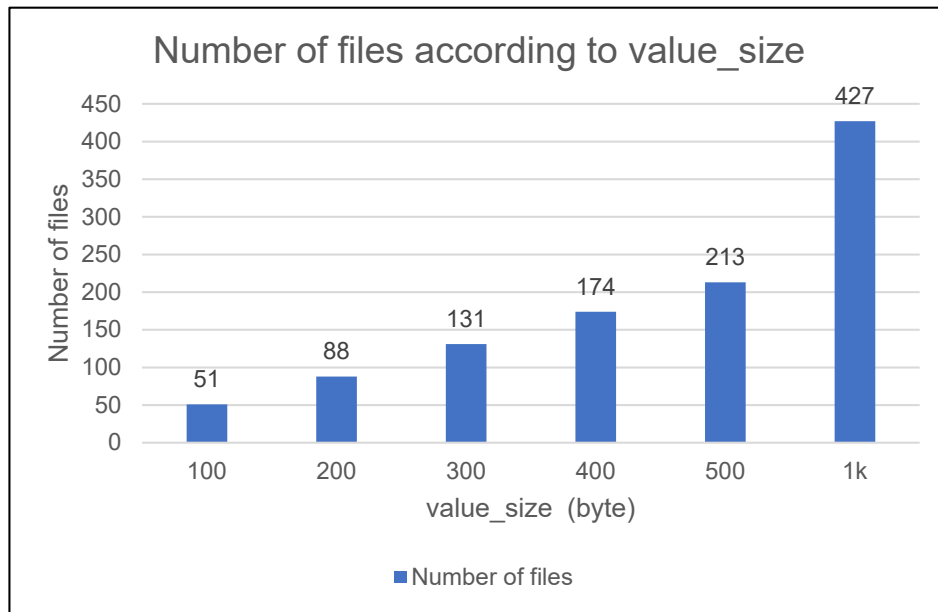
```
solid@sanghyun17-7666596bb6-19j18:~/leveldb_study/leveldb_release/build$ ./db_bench --benchmarks=
"fillseq" --value_size=2000 --compression_ratio=1 --use_existing_db=0
LevelDB: version 1.23
Date: Fri Jul 22 15:54:59 2022
CPU: 40 * Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz
CPU Cache: 14080 KB
Keys: 16 bytes each
Values: 2000 bytes each (2000 bytes after compression)
Entries: 1000000
RawSize: 1922.6 MB (estimated)
FileSize: 1922.6 MB (estimated)
WARNING: Snappy compression is not enabled
-----
fillseq : 10.353 micros/op; 185.7 MB/s
```

3. Further research

1. Figure out more specific format of each blocks in SSTable
2. Figure out the process(functions) of generating each blocks
(From Source code level)

Appendix

- How many files will be generated more?



<Number and Size of files generated>

Appendix – Experiment info

- <Environment 1>
- CPU : 4*Intel® Core™ i5-6600 CPU @ 3.3GHz
- CPUCache : 6144KB
- SSD : Samsung SSD850 500GB
- <Environment 2>
- CPU : 40*Intel® Xeon® Silver 4210R CPU @2.40GHz
- CPUCache : 14080KB

Thank you