Report of Project 1

Kai Li

Abstract

Nowadays, LevelDB has become a very popular key-store database prototype. In this project, we demonstarte on how to investigate LevelDB by running some benchmarks to show the performance under different configurations. We tune three configurable parameters and report the results, by comparing the results, we further explain how and why each option influences the system's performance.

1 Introduction

In order to understand the properties of LevelDB [1] and how each configureable knob influences the system's performance, a scientific way to study it is to instrument it and conduct experiments to do the comparion by changing those knobs. To do so, first of all, we need to clone the source code from Github and build the binaries from the source code. By following the README file, you are supposed to build the binaries, including some built-in benchmark tools such as db_bench, db_test, skiplist_test, etc. Once you have built everything correctly, you can locate the binaries in your specified building folder. Then the next step is to run those banchmark tools to get a sense of how the output format looks like. In this report, we picked db_bench to run the experiments. Of course there are other benchmarks (e.g., db_test, skiplist_test), feel free to test it at your preference.

Secondly, once we understand how to run the benchmarks and knew the printed metrics, we can change some of the options when running the tools. The default parameter would be taken if not specified, For example, in db_bench.cc, we found there are about 16 tunable knobs that take inputs from the command line. For this project report, we selected 3 options that we are instereted in and summarized them in Table 1. We can specify different values in the comand line and then execute it to get a new experiment result, by comparing the results of different specified values, we can infer how a particular option influences the system's performance. In each experiment, we only pick up one option, and repeat the aforementioned process, we can infer how each option affects the system's performance metrics.

Options	***Description***	***Default value***
block_size	Approximate size of user data packed per block before compression.	4096 bytes
write_buffer_size	Number of bytes to buffer in memtable before compacting	4 MB
value_size	Size of each value	100 bytes

Table 1: Options in LevelDB

The remainder of this report are orginized as follows, in section 2, we elaborate on the experiment methodology. Section 3 discusses and evaluates the experiment results. Section 4 concludes the project report.

2 Experimental Methodology

2.1 Experimental Environment

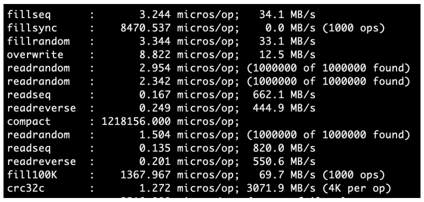
For this project, we conducted the experiments in an machine equipped with Intel 4-core i5-7500 CPU of 3.40 GHz with 8 MB cache, a 8 GB RAM and 256 GB disk. We run the workload speficied in db_bench. We are curioused about the following options, *block_size*, *write_buffer_size*, *value_size* and believe these options are criticl to understand the performance. For instance, we suspect *block_size* could affect write performance as this is the granularity of write operations, large block size may increase the write amplification. For these three options, we assigned it with three different values and run the experiments and understand the options' effect by comparing the results.

2.2 Iterate each option

1. Option *block_size* is set to be 1/4/16 KB in three experiments respectively. The result is presented in Figure 1.

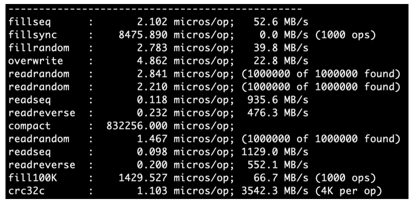
2. Option *write_buffer_size* is set to 1/4/16 MB in three experiments respectively. The result is presented in Figure 2.

3. Option *value_size* is set to be 10/20/40 bytes in the three experiments respectively. The experiment results are shown presented 3.



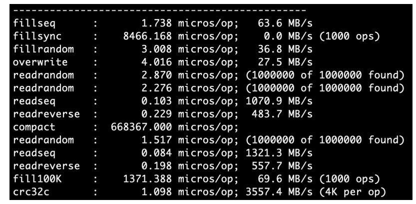
```
fillseq      :       3.244 micros/op;   34.1 MB/s
fillsync     :    8470.537 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       3.344 micros/op;   33.1 MB/s
overwrite    :       8.822 micros/op;   12.5 MB/s
readrandom   :       2.954 micros/op; (1000000 of 1000000 found)
readrandom   :       2.342 micros/op; (1000000 of 1000000 found)
readseq      :       0.167 micros/op;  662.1 MB/s
readreverse  :       0.249 micros/op;  444.9 MB/s
compact      : 1218156.000 micros/op;
readrandom   :       1.504 micros/op; (1000000 of 1000000 found)
readseq      :       0.135 micros/op;  820.0 MB/s
readreverse  :       0.201 micros/op;  550.6 MB/s
fill100K     :    1367.967 micros/op;   69.7 MB/s (1000 ops)
crc32c       :       1.272 micros/op; 3071.9 MB/s (4K per op)
```

(a) block size = 1 KB



```
------------------------------------------------
fillseq      :       2.102 micros/op;   52.6 MB/s
fillsync     :    8475.890 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       2.783 micros/op;   39.8 MB/s
overwrite    :       4.862 micros/op;   22.8 MB/s
readrandom   :       2.841 micros/op; (1000000 of 1000000 found)
readrandom   :       2.210 micros/op; (1000000 of 1000000 found)
readseq      :       0.118 micros/op;  935.6 MB/s
readreverse  :       0.232 micros/op;  476.3 MB/s
compact      :  832256.000 micros/op;
readrandom   :       1.467 micros/op; (1000000 of 1000000 found)
readseq      :       0.098 micros/op; 1129.0 MB/s
readreverse  :       0.200 micros/op;  552.1 MB/s
fill100K     :    1429.527 micros/op;   66.7 MB/s (1000 ops)
crc32c       :       1.103 micros/op; 3542.3 MB/s (4K per op)
```

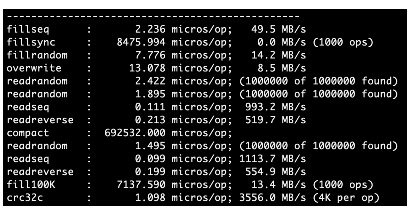
(b) block size = 4 KB



```
------------------------------------------------
fillseq      :       1.738 micros/op;   63.6 MB/s
fillsync     :    8466.168 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       3.008 micros/op;   36.8 MB/s
overwrite    :       4.016 micros/op;   27.5 MB/s
readrandom   :       2.870 micros/op; (1000000 of 1000000 found)
readrandom   :       2.276 micros/op; (1000000 of 1000000 found)
readseq      :       0.103 micros/op; 1070.9 MB/s
readreverse  :       0.229 micros/op;  483.7 MB/s
compact      :  668367.000 micros/op;
readrandom   :       1.517 micros/op; (1000000 of 1000000 found)
readseq      :       0.084 micros/op; 1321.3 MB/s
readreverse  :       0.198 micros/op;  557.7 MB/s
fill100K     :    1371.388 micros/op;   69.6 MB/s (1000 ops)
crc32c       :       1.098 micros/op; 3557.4 MB/s (4K per op)
```

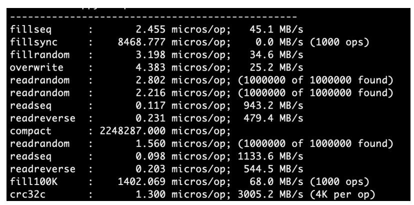
(c) block size = 16 KB

Figure 1: Experiment results of changing block_size



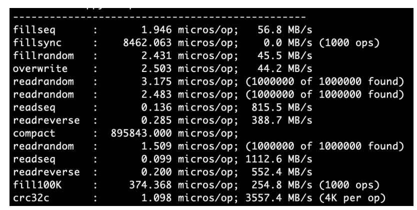
```
------------------------------------------------
fillseq      :       2.236 micros/op;   49.5 MB/s
fillsync     :    8475.994 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       7.776 micros/op;   14.2 MB/s
overwrite    :      13.078 micros/op;    8.5 MB/s
readrandom   :       2.422 micros/op; (1000000 of 1000000 found)
readrandom   :       1.895 micros/op; (1000000 of 1000000 found)
readseq      :       0.111 micros/op;  993.2 MB/s
readreverse  :       0.213 micros/op;  519.7 MB/s
compact      :  692532.000 micros/op;
readrandom   :       1.495 micros/op; (1000000 of 1000000 found)
readseq      :       0.099 micros/op; 1113.7 MB/s
readreverse  :       0.199 micros/op;  554.9 MB/s
fill100K     :    7137.590 micros/op;   13.4 MB/s (1000 ops)
crc32c       :       1.098 micros/op; 3556.0 MB/s (4K per op)
```

(a) write buffer size = 1 MB



```
------------------------------------------------
fillseq      :       2.455 micros/op;   45.1 MB/s
fillsync     :    8468.777 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       3.198 micros/op;   34.6 MB/s
overwrite    :       4.383 micros/op;   25.2 MB/s
readrandom   :       2.802 micros/op; (1000000 of 1000000 found)
readrandom   :       2.216 micros/op; (1000000 of 1000000 found)
readseq      :       0.117 micros/op;  943.2 MB/s
readreverse  :       0.231 micros/op;  479.4 MB/s
compact      : 2248287.000 micros/op;
readrandom   :       1.560 micros/op; (1000000 of 1000000 found)
readseq      :       0.098 micros/op; 1133.6 MB/s
readreverse  :       0.203 micros/op;  544.5 MB/s
fill100K     :    1402.069 micros/op;   68.0 MB/s (1000 ops)
crc32c       :       1.300 micros/op; 3005.2 MB/s (4K per op)
```

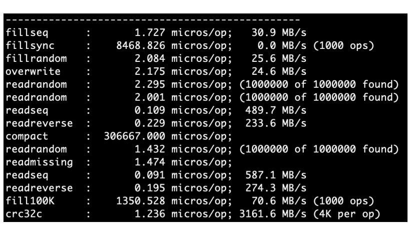
(b) write buffer size = 4 MB



```
------------------------------------------------
fillseq      :       1.946 micros/op;   56.8 MB/s
fillsync     :    8462.063 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       2.431 micros/op;   45.5 MB/s
overwrite    :       2.503 micros/op;   44.2 MB/s
readrandom   :       3.175 micros/op; (1000000 of 1000000 found)
readrandom   :       2.483 micros/op; (1000000 of 1000000 found)
readseq      :       0.136 micros/op;  815.5 MB/s
readreverse  :       0.285 micros/op;  388.7 MB/s
compact      :  895843.000 micros/op;
readrandom   :       1.509 micros/op; (1000000 of 1000000 found)
readseq      :       0.099 micros/op; 1112.6 MB/s
readreverse  :       0.200 micros/op;  552.4 MB/s
fill100K     :     374.368 micros/op;  254.8 MB/s (1000 ops)
crc32c       :       1.098 micros/op; 3557.4 MB/s (4K per op)
```

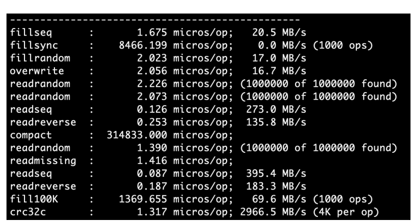
(c) write buffer size = 16 MB

Figure 2: Experiment results of changing write_buffer_size



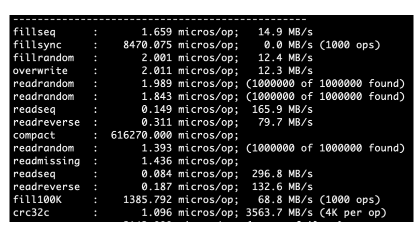
```
------------------------------------------------
fillseq      :       1.727 micros/op;   30.9 MB/s
fillsync     :    8468.826 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       2.084 micros/op;   25.6 MB/s
overwrite    :       2.175 micros/op;   24.6 MB/s
readrandom   :       2.295 micros/op; (1000000 of 1000000 found)
readrandom   :       2.001 micros/op; (1000000 of 1000000 found)
readseq      :       0.109 micros/op;  489.7 MB/s
readreverse  :       0.229 micros/op;  233.6 MB/s
compact      :  306667.000 micros/op;
readrandom   :       1.432 micros/op; (1000000 of 1000000 found)
readmissing  :       1.474 micros/op;
readseq      :       0.091 micros/op;  587.1 MB/s
readreverse  :       0.195 micros/op;  274.3 MB/s
fill100K     :    1350.528 micros/op;   70.6 MB/s (1000 ops)
crc32c       :       1.236 micros/op; 3161.6 MB/s (4K per op)
```

(a) value size = 10 bytes



```
------------------------------------------------
fillseq      :       1.675 micros/op;   20.5 MB/s
fillsync     :    8466.199 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       2.023 micros/op;   17.0 MB/s
overwrite    :       2.056 micros/op;   16.7 MB/s
readrandom   :       2.226 micros/op; (1000000 of 1000000 found)
readrandom   :       2.073 micros/op; (1000000 of 1000000 found)
readseq      :       0.126 micros/op;  273.0 MB/s
readreverse  :       0.253 micros/op;  135.8 MB/s
compact      :  314833.000 micros/op;
readrandom   :       1.390 micros/op; (1000000 of 1000000 found)
readmissing  :       1.416 micros/op;
readseq      :       0.087 micros/op;  395.4 MB/s
readreverse  :       0.187 micros/op;  183.3 MB/s
fill100K     :    1369.655 micros/op;   69.6 MB/s (1000 ops)
crc32c       :       1.317 micros/op; 2966.5 MB/s (4K per op)
```

(b) value size = 20 bytes



```
------------------------------------------------
fillseq      :       1.659 micros/op;   14.9 MB/s
fillsync     :    8470.075 micros/op;    0.0 MB/s (1000 ops)
fillrandom   :       2.001 micros/op;   12.4 MB/s
overwrite    :       2.011 micros/op;   12.3 MB/s
readrandom   :       1.989 micros/op; (1000000 of 1000000 found)
readrandom   :       1.843 micros/op; (1000000 of 1000000 found)
readseq      :       0.149 micros/op;  165.9 MB/s
readreverse  :       0.311 micros/op;   79.7 MB/s
compact      :  616270.000 micros/op;
readrandom   :       1.393 micros/op; (1000000 of 1000000 found)
readmissing  :       1.436 micros/op;
readseq      :       0.084 micros/op;  296.8 MB/s
readreverse  :       0.187 micros/op;  132.6 MB/s
fill100K     :    1385.792 micros/op;   68.8 MB/s (1000 ops)
crc32c       :       1.096 micros/op; 3563.7 MB/s (4K per op)
```

(c) value size = 40 bytes

Figure 3: Experiment results of changing value_size

3 Evaluation Results

In this section, we evaluate the experiment results of each option.

In Figure 1, the *block_size* is varied from 1 KB, 4 KB, to 16 KB. The performance metrics printed by *db_bench* shows that write-related metrics are affected by this options, for example, with the increased block size, overhead of sequential write in asynchrous mode (*fillseq*) is reduced from 3.244 us to 2.102 us, and overhead of random write in in asynchrous mode (*fillrandom*) is reduced from 3.334 us to 2.783 us. However, read-related metrics are not affected. Therefore, based on the result, we can tell that the larger block size it is, the smaller overhead of write operations in in asynchrous mode LevelDB wil have.

In Figure 2, the *write_buffer_size* is varied from 1 MB, 4 MB, to 16 MB. It can be seen that write-related metrics are influenced, including *fillseq*, *fillrandom*, *overwrite*,etc. This observation can be explained by that large write buffer size can reduce the frequency of disk operations when encountered write opeations. However, increasing write buffer size has the side-effect of increasing the overhead of read-related operations, for instance, random read (*readrandom*), sequential read (*readseq*), reverse read (*readreverse*) all are slightly increased when the write buffer size is enlarged.

In Figure 3, the *value_size* is varied from 10 bytes, 20 bytes, to 40 bytes. It can be seen that the metric of compaction is affected by this option, with the growing of *value_size*, the overhead of compaction operation is increased as well, which grows from 306 ms to 616 ms eventually. This observation can be explained by that large value size makes compaction process take more effort to compress the object.

4 Conclusion

In this report, we investigated the options provided by LevelDB. To understand how each option affects the system's performance, we selected 3 options, *block_size*, *write_buffer_size* and *value_size* and conducted experiments by changing each option to different values to do the comparison, the results show that *block_size* affects write performance while not affect read performance, similarly, *write_buffer_size* affects write performace, it also slightly affects read performance. Finally, *value_size* affects the compaction overhead.

References

[1] Google. Leveldb. `https://github.com/google/leveldb`.