Report of Lab Project 2: WiscKey

1001556282, Xiaofeng Wu

1. Design Principle

1.1 Overview

The key idea of WiscKey is to separate keys and values: keys and address of value are stored in levelDB, while values are stored in SSD. When a user queries for a key, first we look up the key in the LSM-tree; when we found it, then the its address is retrieved. Next, WiscKey gets the value from the logfile.

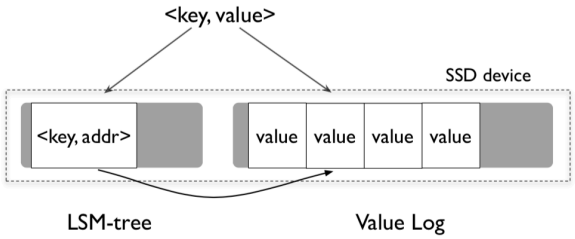

Figure 1: Separation of <key, value>
(originally from paper)

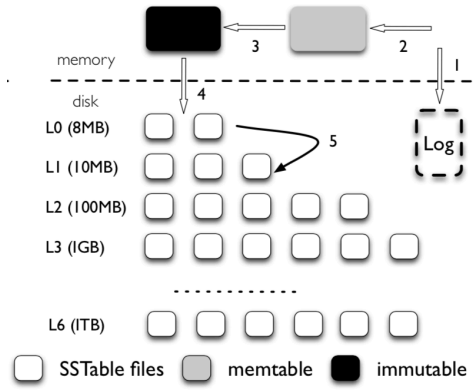

Figure 2: <key, address> is in levelDB; <address, value> is in log file
(originally from paper)

1.2 Advantages

(adopted and cited from the original paper AS the paper clearly and excellently articulate these parts)

1.2.1 Compaction only needs to sort keys, while values can be managed separately.

For the reason that keys are usually smaller than values, compacting only keys could significantly reduce the amount of data needed during the sorting. For WiscKey, only the location of the value is stored in the LSM-tree with the key, while the actual values are stored elsewhere in an SSD-friendly fashion. With this design, for a database with a given size, the size of the LSM-tree of WiscKey is much smaller than that of LevelDB.

1.2.2 The smaller LSM-tree can remarkably reduce the write amplification for modern workloads that have a moderately large value size.

1.2.3 WiscKey's smaller read amplification improves lookup performance. Since the LSM-tree of WiscKey is much smaller than LevelDB (for the same database size), a lookup may search fewer levels of table files in the LSM-tree and a significant portion of the LSM-tree can be easily cached in memory. Hence, each lookup only requires a single random read (for retrieving the value) and thus achieves a lookup performance better than LevelDB.

2. Implementation

I briefly describe the idea of the implementation below.

2.1 open_wisckey
We open leveldb first. Then, we create a logfile.txt to store <addr, value>. If logfile.txt does not exist, we will create it; otherwise we just open it. Next we assign the file handler to wk->logfile.

2.2 close_wisckey
We delete leveldb, close logfile.txt, and finally delete pointer wk.

2.3 wisckey_set
We use leveldb_set to store <key, addr> first.
The addr here is a combination of address of the first character of the value plus a space, and the offset. Next, we append the value to the logfile.txt.

2.4 wisckey_get
We use leveldb_get to retrive addr first. Then we use addr to go to first character of the value, and read the value by its length from logfile.txt.

2.5 wisckey_del
We only delete <key, addr> in the leveldb but don't delete content in the logfile.txt as mentioned in the paper. We leave values that are not deleted to be garbage collected.

3. Evaluation

We compare WiscKey with baseline(levelDB) as follows:

3.1 set key and value

time(sec) / value size	WiscKey	levelDB
1000	8.17	68.32
2000	3.68	56.36

3.2 get values by keys sequentially

time(sec) / value size	WiscKey	levelDB
1000	5.37	7.95
2000	3.05	3.60

3.3 get values by keys randomly

time(sec) / value size	WiscKey	levelDB
1000	5.68	7.14
2000	2.46	3.40

3.4 delete all <key, value> pairs

time(sec) / value size	WiscKey	levelDB
1000	4.40	9.00
2000	1.55	4.40