Paper Review:
"*The Design of a File System for Key-value SSDs*"

Seri Lee
Seoul National University
sally20921@snu.ac.kr

	KV SSD	Block SSD
Access unit	Object	Fixed sector
Addressing	Key (∼255B, string)	Logical Block Address (LBA, 64 bits)
Primitive Operation	SET() GET() DELETE() ITERATOR()	WRITE() READ() TRIM() SEEK()
Atomicity	Object-level	Sector-level
Transaction	Supported (using compound command)	Not Supported

Table I
COMPARISON OF BLOCK SSD AND KV SSD INTERFACES

***Abstract*—**

***Index Terms*—**

I. KV SSD

Each object contains data which are often referred to as value. Object data are stored and retrieved using a key which uniquely identifies the value. Unlike the typical block SSD, KV SSD supports variable-size objects (not fixed to 4KB), atomicity of one or multiple KV operations using compound command, and richer operations. As some might notice, the KV interface can be though of as the general form of the block I/O interface. Internally, KV SSD writes all the data in an append-only manner like the write-ahead logging (WAL).

II. CHALLENGES OF DESIGNING A FILE SYSTEM ON TOP OF KV SSD

A. File I/O Translation

One technical issue is the efficient translation of file I/O commands to KV ones. KVFS receives a file I/O request from the virtual file system and translates it into KV commands, which are then transferred to KV SSD. Before delving into details, we would like to emphasize that additional in-memory or on-disk data structures are unnecessary for file translation.

Creating File and Directory

Two SET()s and two GET()s are necessary. It is worth noting that two SET() operations can be performed atomically by grouping them into compound command. Suppose that '/foo/bar.txt' is newly created. KVFS first checks the existence of the directory, and then sees if the same file was already created. Two SET()s are then followed to create a new meta object and update the access time of the parent directory.

Writing Data

KVFS first checks the existence of a file to access. Then, it gets the start offset of the file and the length of bytes to write from the parameters of the write() call. KVFS obtains suffix numbers of sub-objects. KVFS then writes new sub-object to KV SSD via SET(). KVFS updates the meta object for the file. The meta object and sub-objects can

be updated atomically if they are grouped together with compound command.

REFERENCES

[1]
[2]