Sayed Ismail Safwat (**saysa289**)

A new File System for Flash Storage

Introduction

File systems is a collection of files maintained on a disk drive or a partition, in which file systems are used to handle the data management of the storage. There are many formats designed in Linux file system over time; in this paper the F2FS (Flash-friendly File System) file system is going to be discussed.

Over time and with storage needs growth, the need of multiple flash chips via a single controller increased as well, the controller maintains a flat translation layer (FTL) to track the valid data of NAND flash memory. However, this "solution" can lead to some odd scenarios e.g. random writes which causes internal fragmentation. The main problem is that the other file system formats are able to tackle the aforementioned scenarios, but they are not designed without taking into account the modern flash storage devices which as a result can effect the device lifetime and performance.

F2FS is designed on append-only logging with the modern storage devices in mind. The file system has shown high performance especially in modern mobile systems, beating the previous formats like EXT4, BTRFS.

Design and Implementation of F2FS

Disk Layout

First things first, the NAND flash memory is considered when designing the F2FS on-disk data structures. It consists of three divisions: zone, section, segment, where segment is the smallest unit of management in F2FS, section contains the segments and the zone includes the section. These units plays a big role in cleaning and logging. The segment area which are all fixed-sized are divided into following six fragments: Superblock(SB), Checkpoint(CP), Segment Information Table(SIT), Node Address Table(NAT), Segment Summary Area(SSA) and Main Area.

File Structure

F2FS is similar to LFS but differs when it comes to handling the new flash storage devices, meaning that F2FS maintains a structure extending the inode map to find indexing blocks by using their node ID. It addresses the "wandering tree problem" by updating only one direct node block and the NAT entry, while in LFS the direct and indirect blocks are updated.

Directory Structure

A directory entry is comprised of hash value of the file name, inode number, the length of the file name and the file type (e.g. directory, symbolic link). A directory entry block (called dentry) is composed of arrays of slots and names including a bitmap which determines if a dentry is valid or not. Directory structure in F2FS is implemented on a multi-level hash tables wherein each level consists of a hash table with a fixed number of hash buckets. In order to find a file name, first a hash value of the file is calculated, then it keeps scanning the hash table in each level to find the corresponding dentry that includes the name and inode of the file.

Multi-head Logging

F2FS has support for multi-head logs for hot and cold data separation, which means that it uses six log areas in the Main Area segment to manage the hot and cold data separation. Node blocks and Data blocks are classified as hot, warm and cold depending on their level of usage (read/write) frequency.

Cleaning

In order to secure free segments, F2FS executes cleaning process which can be done in two ways: Foreground and background. The foreground cleaning is executed on demand meaning that it is processed only when there are not free sections, while the background is implemented periodically. Cleaning process is done in three steps: Victim selection, Valid block identification and migration and Post-cleaning process.
In the victim selection step, there are two methods for cleaning — greedy and cost-benefit. F2FS file system uses the greedy algorithm for its foreground cleaning and uses the cost-benefit algorithm for its background cleaning.
For valid identification step, F2FS uses a bitmap, where each bit determines the validity of the block and in case they were valid they will be migrated.

✺ Adaptive Logging

There are two logging policies — normal logging and threaded logging which can be helpful in block allocation efficiency. In the traditional LFS-based file systems, all modifications are written sequentially to the disk which contributes to a better performance in writing and cash recovery.
F2FS maintains both policies and switches between them depending on the file system status.
In normal logging blocks are written to clean segments whereas in threaded logging blocks are written in dirty segments which in short words is turning random writes into sequential writes.

✺ Checkpointing and Recovery

F2FS uses the checkpointing model to maintain the file system consistency. In order to implement it, the file system executes the checkpointing model which is a procedure of following: flushing the dirty nodes and dentry blocks in the page cache, canceling the system calls (e.g. create, unlink), writing metadata, NAT, SIT and SSA to its corresponding area. At the end, a checkpoint pack which is composed of header and footer, NAT and SIT bitmaps, NAT and SIT journals, Summary blocks of active segments and Orphan blocks is written to the CP area.
This way F2FS manages to handle case scenarios such as power outage or system crashes by rolling back to the latest checkpoint it had created.

✺ Evaluation

Experimental Setup

In general, the results shows the dominance of F2FS in the experiment.
The file system is tested on two different target systems — mobile system and server system. Both systems run the kernel versions 3.4.5 and 3.14 respectively. In the experiment, F2FS is compared with EXT4, BTRFS (based on copy-on-write file system) and NILFS2(based on LFS file system). Mobibench is run to measure SQLite performance while Filebench is maintained to measure the server performance.
In mobile system case, the results show significant difference in performance in RW (random writes) in comparison to other FS. The same difference is evaluated in SQLite performance.
In server system case, there is no sign of difference in F2FS performance in video-server. However, it shows a 2.4x performance gain in fileserver due its background cleaning algorithm,— a function EXT4 lacks.

✺ Related Work

Since F2FS is implemented based on LFS, it is important to go through how LFS has evolved over time. Rosenblum et al. Wilkes et al. suggested their proposal of transferring valid blocks of a victim segment to holes. Later on Matthews et al. suggested adaptive cleaning methods — choosing an algorithm depending cost-benefit evaluation.

As discussed before, NAND flash memory has been used previously but in a raw format meaning that the file system had a direct access. On the other hand F2FS maintains the flash memory via a FTL controller.

File systems have had different type of approaches when it comes to FTL optimizations. For example DAC file system harnesses a page-mapping FTL that collects data by monitoring access at run time, thus it leads to larger page mapping tables. To avoid this, DFTL solves it by loading a portion of the page map into working memory and offering the RW (random-write) benefits of page mapping for devices with less RAM.

Conclusion

After studying and evaluating the paper, it shows that the F2FS is arguably one of the most optimal file systems that can be used in the both mobile and server systems. However, I believe with the rapid evolution in both grounds — software and hardware, F2FS is not fairly adopted by the OEMs even though it's not too old.
This thought leads to my question that (1)“How can they make it to being accepted and adopted widely by OEMs in the future?”
(2) “What are the chances of F2FS to fail because of its small audience?”
(3) “With the flash storage getting faster and faster, how can F2FS cope with that speed?”

References

Lee, Changman. Sim, Dongho. Hwang, Joo-Young. Cho, Sangyeun (2015) ‘F2FS: A New File System for Flash Storage’, 13th USENIX Conference on File and Storage Technologies.