F2FS: A New File System for Flash Storage

`Log-structured` `FileSystem` `Flash` `SSD` `FAST15`

Lee, C., Sim, D., Hwang, J. Y., & Cho, S. (2015, February). F2FS: A New File System for Flash Storage. In FAST (pp. 273-286).

背景

NAND flash具有数据异地更新、读写擦除性能不对称和擦除次数有限的特性。观察发现，对SSD频繁的随机写操作会导致其内部空间碎片化严重，不能充分发挥SSD的性能。日志文件系统能较好解决随机写的问题，本文提出F2FS,结合NAND flash特性进行日志文件系统的设计。

主要思路以及贡献

1. 闪存友好的盘上布局（Flash-friendly on-disk layout）

- 闪存Blocks组成segment,连续的segments组成section, sections组成zone。F2FS以segment(闪存block对齐)为单位进行空间分配，以section为单位进行cleaning操作。

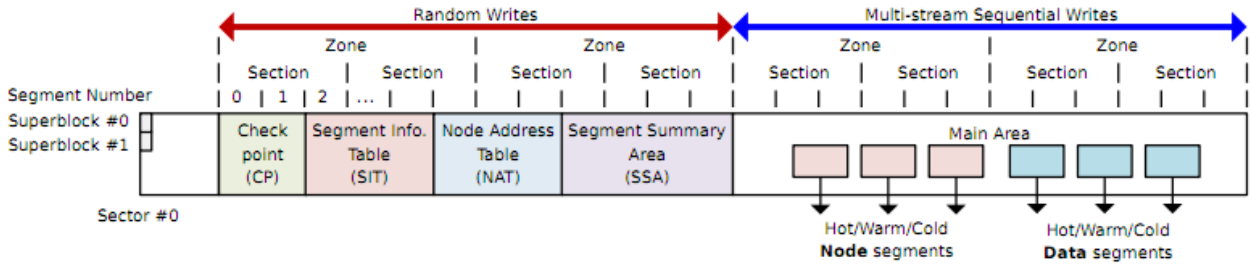

Figure 1: On-disk layout of F2FS.

  - Main area由node和data组成。node包括inode或数据块的索引，data为文件（目录和用户文件）数据。某个Section只会存储node或data中的一类。
  - /dir/file查询操作与ext3，LFS等很类似（Super-block(root-inode)-->root-block(dir-inode)-->dir-block(file-inode)-->file-block），不过使用了NAT，后面会

叙述。

2. 有效的索引结构（Cost-effective index structure）

- 通过引入NAT(node address table)来索引间接块位置，代替传统的间接指针方法，以消除“wandering tree”问题（由于间接指针的存在，某个数据更新可能带来一连串的数据更新操作）。
  - NAT (nodeID,block address),间接块中存储nodeID，而不是块地址。因而在大文件操作时，LFS不仅需要更新直接块指针还要更新多级的间接块。F2FS则只需更新直接块。

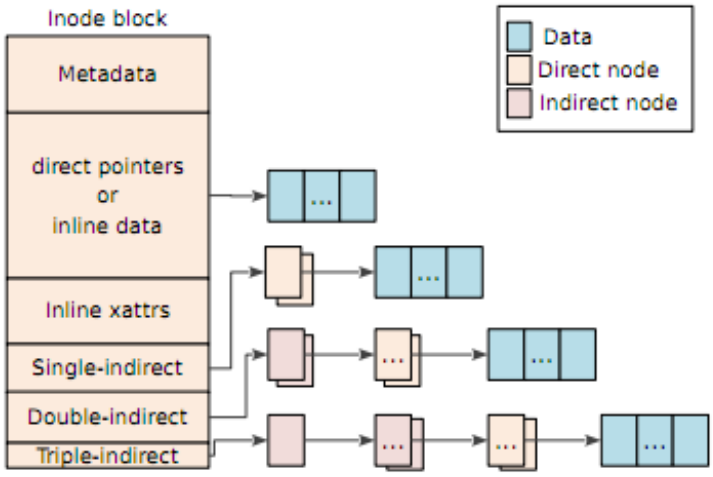

Figure 2: File structure of F2FS.

3. 多日志操作（Multi-head logging）

- 设计了一个高效的日志操作时执行的冷热数据区分机制；维护多个日志头和日志段，同时进行多个日志段的操作，将数据和元数据根据不同的更新频率（热度）分布到不同的日志段上。
  - F2FS静态地将数据分为三种不同的热度，并且分别为Node和Data部分的热度做了区分。F2FS默认同时使用使用6个log,对应热度和数据类型，但也支持灵活的配置。
  - FTL可以进行zone空间的配置，结构不同的log区，避免不同数据的混合布局，以协调降低GC开销。

Type	Temp.	Objects
Node	Hot	Direct node blocks for directories
	Warm	Direct node blocks for regular files
	Cold	Indirect node blocks
Data	Hot	Directory entry blocks
	Warm	Data blocks made by users
	Cold	Data blocks moved by cleaning; Cold data blocks specified by users; Multimedia file data

4. 自适应的日志操作（Adaptive logging）

- 在存储使用率较高时，会采用threaded logging[23]的形式来避免大的写延迟。
  - 正常情况下，数据块以顺序写的形式写到干净的段中。随着空闲空间的减少，cleaning的开销增加，性能降低。此时采取threaded logging的形式在可以写的地方进行数据写，而放弃顺序写的思路。

5. 基于fsync加速的roll-forward恢复机制

- 通过最小化所需的元数据写操作数量和设计一种高效的roll-forward恢复同步写操作的机制，减少fsync请求的开销，达到优化小的同步写操作的目的。
  - 一些应用如数据库（SQLite）经常性地对文件进行小数据的更新，并且执行fsync来进行缓存和存储设备数据的同步。简单的实现会在fsync时触发checkpoint操作，性能较差。主要思路是，fsync时，仅仅写数据块和与其直接相关的node块，而不写其他的元数据块，并在node中设计特别的标志位。恢复时，在上一次稳定的checkpoint（位置N）和具有特别标志的node(位置N+n)的基础上进行roll-forward的数据恢复。

测试以及结论

- 与EXT4, BTRFS(copy-on-write system),NILFS2(LFS)进行比较，所用的benchmark包括Mobile和Server两个方面。与EXT4比较: Mobile方面，3.1x(iozone)，2x(SQLite); 在Server方面，2.5x（SATA SSD),1.8x(PCIe SSD)

Table 3: Summary of benchmarks.

Target	Name	Workload	Files	File size	Threads	R/W	fsync
Mobile	iozone	Sequential and random read/write	1	1G	1	50/50	N
	SQLite	Random writes with frequent fsync	2	3.3MB	1	0/100	Y
	Facebook-app	Random writes with frequent fsync	579	852KB	1	1/99	Y
	Twitter-app	generated by the given system call traces	177	3.3MB	1	1/99	Y
Server	videoserver	Mostly sequential reads and writes	64	1GB	48	20/80	N
	fileserver	Many large files with random writes	80,000	128KB	50	70/30	N
	varmail	Many small files with frequent fsync	8,000	16KB	16	50/50	Y
	oltp	Large files with random writes and fsync	10	800MB	211	1/99	Y

- F2FS是一个针对闪存存储设备的成熟的Linux文件系统。但是，在当前架构下，对闪存特性的利用还不够彻底，主要是LFS和FTL在GC上会有一些冗余和冲突，因此可以考虑open-channel SSD上的软硬件协同设计。

引用

- Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh. Optimizations of LFS with slack space recycling and
  lazy indirect block update. In Proceedings of the Annual Haifa Experimental Systems Conference,page 2, 2010.