Flash-Aware File System Block Allocation for Mobile Consumer Devices

Hyukjin Kwon*, Dongkun Shin**
*Sungkyunkwan University, Suwon, Korea, hjkwon0124@skku.edu
** Sungkyunkwan University, Suwon, Korea, dongkun@skku.edu

Abstract **-** The performance of recent mobile devices is highly related to storage and file system. However, the current EXT4 file system is not optimized for NAND flash memory. In this paper, we propose a novel flash-aware file system block allocation technique, which reserves several address regions for hot file. The scheme significantly reduces the garbage collection overhead within the flash memory. In experiments with a flash memory simulator, the proposed flash-aware block allocation improved I/O performance by up to 28% for the storage access traces of Android applications.

Keywords: Flash memory, File system.

1 Introduction

One important component of recent smart mobile devices such as smartphones and tablets is NAND flash memory storage. Storage and file system on mobile devices affect the performance of mobile applications if they manage a large number of user files. The user interactivity on web browser application, which has been generally considered to be determined by network performance, is also affected by the storage performance, since it accesses many data files such as cookies and cached files [1].

The storage I/O performance depends on the file system as well as the storage technology. There is a significant difference between the performances of different file systems on a same storage media. Especially, the file system performance on NAND flash memory is determined whether it is designed considering the characteristics of flash memory.

NAND flash memory has different characteristics from hard disk drives. It has no rotational delay and seek time. A NAND flash memory chip is partitioned into blocks, and each block is composed of pages. The basic read/write operation unit is a page, while the basic erase operation unit is a block. Once a page is written, the page is no longer updatable before the resident block is erased. Therefore, the out-of-place update scheme is usually adopted to have the updated data written to some other free pages. Therefore, an address mapping scheme is required, which translates between the logical and physical page numbers. If a page-level mapping is used, the mapping table handling overhead may be significant since the mapping table size could be large. Generally, recent flash memory devices use a zone-based address mapping scheme, where the overall address space is partitioned into several regions and each region is managed by a separated mapping table which can be loaded into the SRAM of flash memory chip. Therefore, the number of map loading overhead depends on the spatial locality of access pattern.

When the system runs short of available free pages, pages containing invalid data must be reclaimed by garbage collection which selects a victim block and moves all the valid pages of the block into other blocks with free space. Since the garbage collection invokes a large number of page copy operations, it is important to reduce garbage collection overhead for high performance.

Linux operating system uses EXT4 [2] as a standard file system. EXT4 is a deeply optimized file system targeting for hard disk drives. In order to reduce the rotational delay and the seek time, EXT4 splits the whole disk space into a number of block groups and tries to locate the data blocks and metadata blocks of a file in the same block group. However, the characteristics of flash memory are not considered in the design of EXT4 file system.

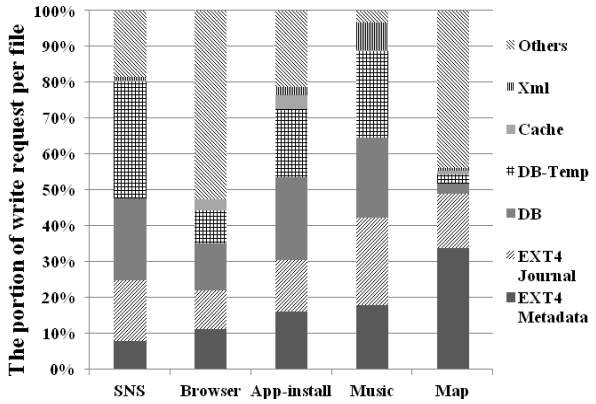
In this paper, we propose the flash-aware file system block allocation technique for EXT4 file system. The technique reduces the garbage collection overhead and mapping table handling overhead by allocating the blocks of frequently-updated files within special regions called hot-zones. The block allocation technique reduces the number of valid pages in a victim flash memory block and the number of map loadings by increasing the spatial locality of hot file blocks.

2 Flash-Aware Block Allocation

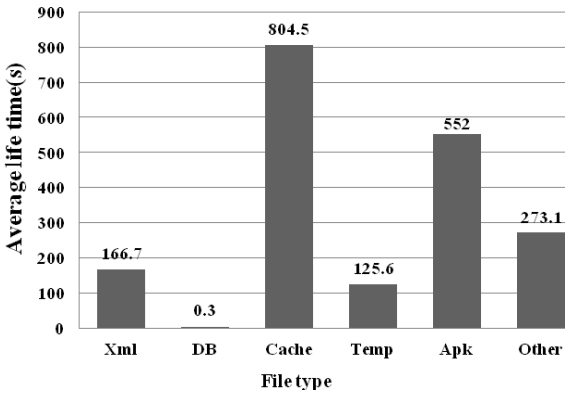
The role of file system block allocator is to assign storage blocks for file data. When a write or read request on a file is sent to the storage device, the request has the logical address information of the target block. The address mapping scheme in storage device determines the

physical page for the request. Therefore, the file system block allocation affects the storage access pattern.

The proposed flash-aware block allocation reserves several address zones for hot files which are frequently updated and thus generate many invalid pages. Therefore, the invalid pages congregate in the flash memory blocks of the reserved zones. In addition, since the lifetime of hot file is short, it is highly probable that hot files share a same block. Then, the garbage collection can find a low-cost victim block in the reserved zone and thus the IO performance is improved.



(a)



(b)

Figure 1. I/O characteristics of Android applications. (a) write requests on different files. (b) average lifetimes of different files

We first investigated the file access patterns of several applications at an Android-based smartphone in order to identify hot files. Figure 1(a) shows which files are frequently updated during the execution of several applications. Except the journal file of EXT4, which uses separated blocks, database files and their metadata are frequently written. Moreover, the lifetime of database file is relatively short as shown in Figure 1(b). Therefore, we determined the database files as hot files.

The next step is to determine the size of a hot zone. The optimal size depends on the internal structure of flash memory chip. Since we cannot look into the flash memory chip, we measured I/O performance while changing the size of hot zone. Form the experiments, we found that 64 MB of size is best.

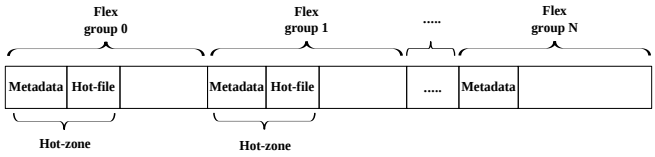


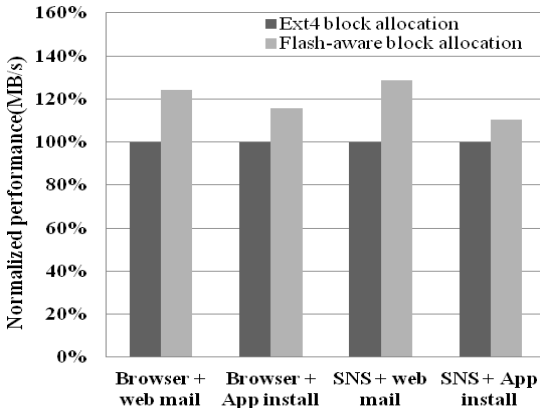
Figure 2. Reserved space for hot-zones

EXT4 file system partitions the whole storage space into several flex groups. The front part of each flex group is reserved for file system metadata such as bitmap table and inode table. Since the metadata are also frequently updated, we reserved 64 MB of hot zone at the front part of each flex group, and allocated the blocks for hot files just after the metadata blocks. Other cold file blocks are allocated from remaining blocks.

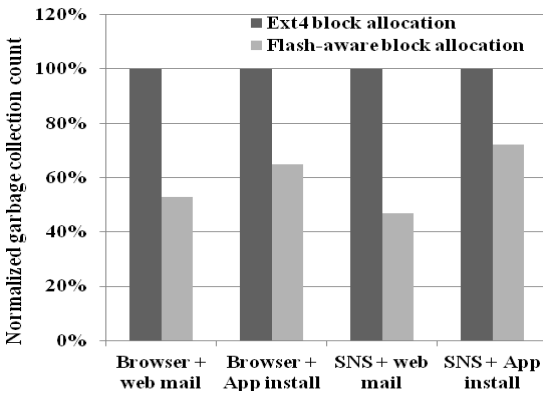
The number of hot zones should be determined considering the total size of hot files, which actually depends on user pattern. We examined ten number of Android-based smartphones used for more than one year so as to know the total size of hot files on average. We found that most of database files are small-sized and their lifetime is short. Therefore, the total sizes of database files are smaller than 90MB on the examined smartphones, and thus four hot zones are sufficient. Among the several hot zones, data blocks and their metadata blocks should be allocated in the same hot zone.

3 Performance Evaluation

We evaluated the flash-aware block allocation with our own flash memory simulator. It implements the page mapping scheme.



(a)



(b)

Figure 3. Evaluation result.
(a) I/O performance. (b) The number of garbage collections

Figure 3(a) compares the I/O performances under different block allocations. The flash-aware block allocation improves I/O performance by 14~28%. This is because the proposed scheme decreases the number of garbage collections as shown in Figure 3(b).

4 Conclusion

In this paper, we proposed the flash-aware block allocation which reserves the address space for hot files. In the experiments, we showed the significant performance improvements by the novel block allocation scheme.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A1A2A10013598).

References

[1] Hyojun Kim, Nitin Agrawal, Cristian Ungureanu, “Revisiting storage for smartphones,” Proceedings of the 10th USENIX conference on File and Storage Technologies, 2012.
[2] Avantika Mathur, et al. “The new EXT4 filesystem: current status and future plans,” Proceedings of the Linux Symposium. Vol. 2. 2007.