A Technique to Enhance Performance of Log-based File Systems for Flash Memory in Embedded Systems

Junkil Ryu and Chanik Park

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
San 31, Pohang, Kyungbuk, Republic of Korea
{lancer, cipark}@postech.ac.kr

Abstract

In this paper, we address the problems of the existing log-based flash memory file systems analytically and propose an efficient log-based file system, which produces higher performance, less memory usage and mount time than the existing log-based file systems. Our ideas are applied to a well-known log-based flash memory file system (YAFFS2) and the performance tests are conducted by comparing our prototype with YAFFS2. The experimental results show that our prototype achieves higher performance, less system memory usage, and faster mounting than YAFFS2, which is better than JFFS2.

1. Motivation

The used areas in the *NAND* flash memory must be scanned to gather the meta-data and data mapping information when the file system is mounted. Thus, the mount time of the file systems is in proportion to the used size of the embedded flash memory. We assume that the file system has a total of n files and each file's size is S pages on average. The read time for the data area and the spare area in a page is denoted by T_D and T_S, respectively. $Pages_{inB}$, T_Blocks, and GC_{th} mean the number of pages in a block, a total number of blocks in the flash memory and garbage collection threshold (a few erased blocks in the flash memory must be reserved for the efficient file updates). T_{yaffs2} is the mount time of YAFFS2. The upper bound of the mount time is because the obsolete pages in the flash memory are not erased yet.

$$n(T_D + T_S) + n \times S \times T_S \;\leq\; T_{yaffs2} \;\leq\; n(T_D + T_S) + T_S \times Pages_{inB} \times (T\_Blocks - GC_{th}) \quad (1)$$

$n \times S \times T_S$ and $T_S \times Pages_{inB} \times (T\_Blocks - GC_{th})$ are overhead to mount the file system. Hence, we introduce the method to reduce these times. In the case of YAFFS2, the entire meta-data and data mapping information in the file system is kept in the system memory. Hence, the system memory usage increases as the flash memory's size increases. When considering YAFFS2's runtime data structure, 1MB system memory per 1GB flash memory (in the case of large-sized (2KB) page) are required for data mapping information and 120B per a file are required (with shortname caching enabled)[1]. YAFFS2's low performance is because the physical address managed in the system memory is not the page address but the group address, which is composed of 16 pages. When reading the wanted page, the number of spare-area read operations to search the page, is 1 in the best case and 16 in the worst case. If seek & read operation starts from the first page in a group, the number of spare-area read operations required to search for the next page is like the following equation (2) when seek distance is even, (3) when odd. a_k is the number of spare-area read operations for k-th seek operation.

$$a_{k+1} = (a_k + upper\_bound(seek\_distance/2))\%16 \quad (2)$$

$$a_{k+2} = (a_k + seek\_distance)\%16. \quad (3)$$

The total overhead of n seek and read operation in the YAFFS2 is as follows:

$$\sum_{k=1}^{n} a_k \quad (4)$$

2. Related Work

To reduce the mount time of the log-based flash memory file systems, Yim et al.[2]'s and Wu et al.[3]'s methods

1-4244-1476-8/07/$25.00

were proposed. [2]'s method is to commit the snapshot of the data structure for the flash memory when the file system is unmounted. This method does not operate well when the file system is not unmounted properly. The old snapshots are not useful in the reconstruction of the file system image in the system memory and the updated areas are not addressed easily. Snapshotting a file system image when unmounting, elongates the shutdown time. To resolve [2]'s problems, [3]'s method was proposed. [3]'s method records the changes of the file system image in the runtime. But when the system crashes, the changed portions in the flash memory cannot be addressed easily because their method records READ/WRITE I/O operations to track the change of the file system image. In Ryu et al.[4], we introduced our method to reduce the log-based flash memory file system's mount time and recovery time for the multimedia files simply.

3. The Proposed Method for Log-based File System

To reduce the mount and recovery time of the log-based flash memory file system, our proposed method introduces the startup area. The startup area is allocated separately from the data area, which is managed by YAFFS2 in our prototype. The startup area is managed by SyncManager. The startup area's size is determined in proportion to the number of the files stored in Flash memory. If the size of the startup area is small for the size of the stored files, the size of the startup area will be increased in the runtime by the garbage collector. Equation (5) is the mount time of the proposed method. S_Blocks is the number of the blocks in the startup area. S_GC_{th} is the garbage collection threshold in the startup area. In the Equation (5), $Pages_{inB} \times (S\_Blocks - S\_GC_{th})$ is approximately equal to $Pages_{inB} \times (T\_Blocks - GC_{th}) \div S$. Hence, as the average size of the files is larger, the mount time in our propose method is smaller.

$$n(T_D + T_S) \leq T_{proposed} \leq n(T_D + T_S) + T_S \times Pages_{inB} \times (S\_Blocks - S\_GC_{th}) \quad (5)$$

[Fig. 1] shows the overview of our proposed method. To track the changes of the file system image, [3] records the changes of I/O operations in a log-segment but the number of log-segments increase as the I/O operations increase, which increases the crash recovery time. However, our proposed method focuses on the changed files. To track the change of the file system image, we introduced the filter function in the YAFFS2 to confirm which file is changed. The changed file is recorded by SyncManager. A log page in the startup area is made for a file, which contains the data address mapping for the corresponding file. When a file is updated, the corresponding log page in the startup area is marked as DIRTY in the tag and the SyncManager holds the pointer of the file object. During idle time, the SyncManager writes the changed files' data address mappings into the re-allocated log pages. If the system crash occurs in the state where the startup area is not synchronized with the YAFFS2 data-area, then the updated portions in the flash memory will be tracked quickly by finding the DIRTY-marked log pages in the startup area. To prevent the startup area from aging quickly, SyncManager reduces I/Os onto the startup area by lazy-updating the changes of a file when the file has been unused for the expected time. To achieve high performance, our proposed method uses a

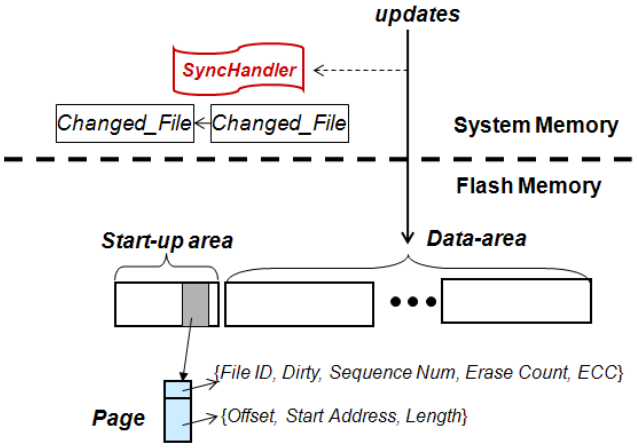

Figure 1. Overview of the proposed method

different page addressing technique from that of YAFFS2. In YAFFS2, the wanted page is found by searching pages in the group containing the wanted page sequentially. To remove this overhead, our proposed method does not use the group address mapping but instead uses the direct page address mapping. Our proposed method uses the B-Tree technique for the data address mapping instead of YAFFS2's Tnode Tree. However, a node in our B-Tree has an entry, which can represent many pages if the pages are stored continuously in the flash memory. A node in our B-Tree represents a logical address, a physical address, and the length of the continuous pages in the flash memory. To control the usage of the system memory, our proposed method unloads the data address mappings of the unused files from the system memory. When the number of the used files exceed the threshold, the unused and old loaded files are unloaded. If a file, which does not has its data mapping in the system memory, is requested by the file system, then our proposed system loads the corresponding file's data address mapping from the startup area in the flash memory, whose address is contained in the file object.

4 Performance Evaluation

The experiments were conducted with YAFFS2 and our prototype, which was implemented based on YAFFS2. Our experimental setup is Intel PXA270A (520MHz), Samsung SDRAM 64MB, and 6x 1GB NAND flash memory. [Fig. 2] shows the flash memory file system's mount time of our prototype and YAFFS2, using mp3 files, whose average file size is 5.9 MB. In [Fig. 2], our prototype's mount time is under 9 seconds but YAFFS2's mount time increases linearly while the size of the data stored in the flash memory increases. It is because YAFFS2 scans the used pages and each block's first page. In the mobile device, our proposed method is efficient and practical because multimedia data are mainly stored, their size is large, and the number of the stored multimedia data is limited. To compare the performance of YAFFS2 and our prototype, we used the iozone benchmark. Our proposed method is a little better than YAFFS2 since our method uses the direct page mapping in the iozone benchmark. [Fig. 3] shows the random seek & 1KB read performance of JFFS2, YAFFS2, vFAT, Ext2, and our prototype. Our proposed method is faster than even vFAT and Ext2 because data mapping information in the proposed method is in the system memory. When the data address mapping is unloaded, the overhead may be introduced to load the data address mapping but the time to read 1 log-page (2KB) is below 0.2 msec.

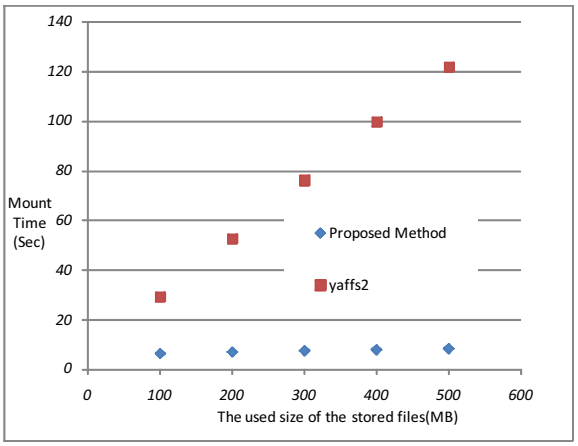

Figure 2. Mount Time vs. The used size of the stored files (average size of the stored files: 5.9 MB)

5 Conclusion

We addressed the problems of the existing log-based flash memory file systems and showed that our method is

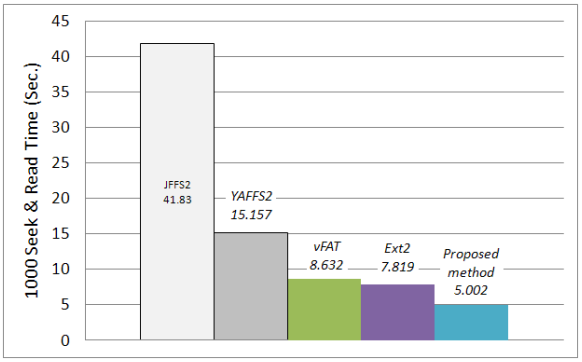

Figure 3. Random Seek & 1KB Read Performance (1000 Times)

better than the existing log-based flash memory file systems in the high performance, fast mount and memory management. This method introduces the start up area allocated separately from the existing file system's data area. A log page in the startup area has the corresponding file in the existing file system and the file's mapping data. When a system crashes, the unsynchronized files are detected easily by confirming the startup area. Thus, the file system mount and crash recovery are fast.

Acknowledgement

This research was supported by the IT R&D program (Development of Wearable Personal Station, 2005-S-065-03) and ITRC (Information Technology Research Center, IITA-2006-C1090-0603-0045) of MIC/IITA.

References

[1] Aleph One Company, "YAFFS: Yet Another Flash File System," *http://www.aleph1.co.uk/yaffs/, 2002.*

[2] Keun Soo Yim, Jihong Kim, and Kern Koh, "A Fast Start-Up Technique for Flash Memory Based Computing Systems," *Proc. of the ACM Symposium on Applied Computing, Santa Fe, USA, March 2005.*

[3] Chin-Hsien Wu, Tei-Wei Kuo, and Li-Pin Chang, "Efficient Initialization and Crash Recovery for Log-based File Systems over Flash Memory," *Proc. of the ACM Symposium on Applied Computing, Dijon, France, April 2006.*

[4] Junkil Ryu and Chanik Park, "Fast Initialization and Memory Management Techniques for Log-Based Flash Memory File Systems," *Proc. of the International Conference on Embedded Software and Systems, Daegu, Korea, May 2007.*