# Fast Initialization and Memory Management Techniques for Log-Based Flash Memory File Systems

Junkil Ryu and Chanik Park

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

**Abstract.** Flash memory's adoption in the mobile devices is increasing for various multimedia services such as audios, videos, and games. The traditional research issues such as out-place update, garbage collection, and wear-leveling are important, the fast initialization and response time issues of flash memory file system are becoming much more important than ever because flash memory capacity is rapidly increasing. In this paper, we propose a fast initialization technique and an efficient memory management technique for fast response time in log-based flash memory file systems. Our prototype is implemented based on a well-known log-based flash memory file system YAFFS2 and the performance tests were conducted by comparing our prototype with YAFFS2. The experimental results show that the proposed initialization technique reduced the initialization time of the log-based flash memory file system regardless of unmounting the file system properly. Moreover our prototype outperforms YAFFS2 in the read I/O operations and the forward/backward seek I/O operations by way of our proposed memory management technique. This technique is also able to be used to control the memory size required for address mapping in flash memory file systems.

**Keywords:** flash memory, log-based file system, file system initialization, high performance, efficient memory management.

## 1 Introduction

Flash memory has been widely adopted as a storage media in the mobile devices because it is non-volitle, shock-resident, and power-economic. However the data page in the flash memory cannot be re-used without a block erase operation and the flash memory's page size is not equal to its block size. Note that I/O operations are performed on a page in the flash memory. Thus, additional functions must be implemented to store files in the flash memory. There are two major approaches to provide the file service in flash memory. One is the native file system approach ([1], [2]) which is aware of the characteristics of the flash memory, that the native file system applies to provide efficient file system service. The other is the block-device emulation approach ([3], [4]) and it creates generic block device environments (translation layer) for the existing file systems to use the flash

Y.-H. Lee et al. (Eds.): ICESS 2007, LNCS 4523, pp. 219–228, 2007.

memory. The two approaches' goal is to have applications accessing data on the flash memory transparently using standard file system APIs. In its early stages, most of the research topics on the flash memory focused on harnessing the flash memory such as out-place update, garbage collection, and wear-leveling. But the performance, reliability, and low power comsumption have become much interested research topics on the flash memory in recent years.

In this paper, the initialization time, performance, and efficient memory management for the flash memory file systems will be handled. When a log-based flash memory file system is mounted, any data areas and all spare areas of the used pages in the flash memory must be scanned to reconstruct its house-keeping data structures (meta-data and data address mapping) in the system memory. This procedure is time consuming while maintaining the stored files' house-keeping data (meta data and data address mapping) in the system memory is memory consuming and impractical because the size of the flash memory is becoming larger and the system memory size in the mobile device does not follow the trend of the flash memory size. [Fig. 1]'s experiment shows that the initialization time of the log-based flash memory file system (YAFFS2) increases linearly as the data stored in the flash memory is increased. [Fig. 2]'s experiment shows that the system memory used for data address mapping and meta-data in YAFFS2 increases as the data stored in the flash memory increases. The experimental setup is as follows: Processor is Intel Burlverde PXA270 (520Mhz), system memory is SAMSUNG SDRAM 64MB, and flash Memory is SAMSUNG 1GB NAND flash memory. Hence, the issues of the initialization time and memory management for the flash memory are very important.

The growth of quality and quantity in the multimedia data has cause the mobile device to store many multimedia data and process the multimedia data encoded in high quality. The existing log-based flash memory file system (ex, YAFFS2) has been focusing on applying the physical characteristics of the flash memory to

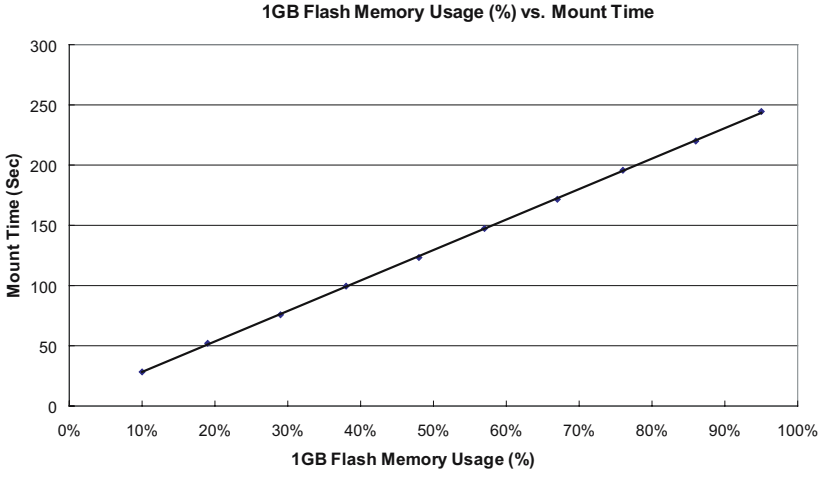


**Fig. 1.** YAFFS2 Mount Time

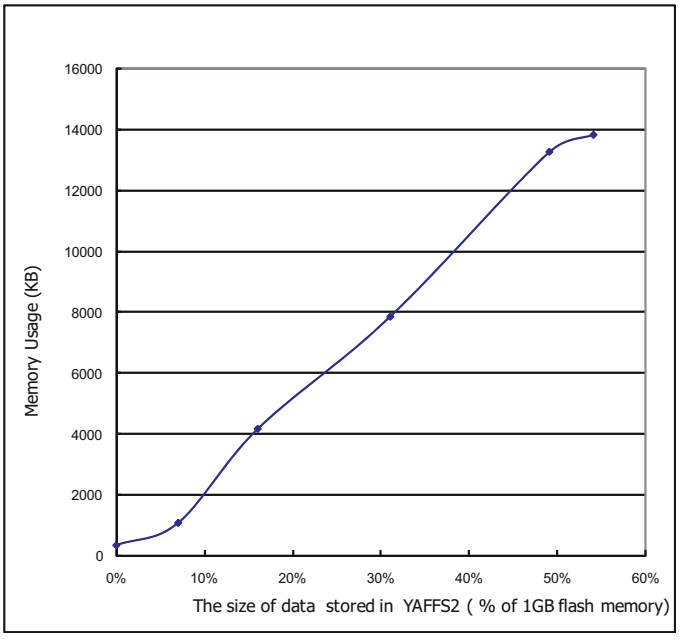


**Fig. 2.** System Memory used for YAFFS2

provide the file system service. Currently, the high performance flash memory file system is needed to process many high quality and quantity multimedia files.

In this paper, a method for fast initialization, high performance and memory management is proposed for flash memory log-based file systems. For the fast initialization, we allocated the startup area seperately in the flash memory, which is managed by SyncManager. For high performance, a page in the flash memory is addressed directly. For the memory management, the B-Tree technique is used instead of Tnode Tree in YAFFS2 and the data addressing mapping can be unloaded in the system memory when it is unused.

The remainder of this paper is organized as follows: Section 2 introduces the related work and motivations. Section 3 introduces the proposed methods and section 4 provices the performance evaluation of the proposed method over YAFFS2. Finally, section 5 draws conclusion.

## 2 Related Work

There are two approaches in the file system implementation to provide data accessing transparently in the flash memory. One is the native flash memory file system and the other is the block device emulation. The existing disk file systems can be used with these block device emulations. This paper discusses the flash memory file system, which manages raw flash memory directly (ex, JAFFS2 and YAFFS/YAFFS2). These flash memory file systems are closely related to

log-structured file system ([7]) because a page in the flash memory cannot be done in-place updates. When updating the data in a page, the data is moved from a page to another page. Unlike disks, we cannot know where the data is. Hence, the data-stored pages must be tracked and managed in the flash memory file system. In order to resolve this problem, the mapping concept of logical address space and physical addresss space is adopted, where the logical address space is indexed by logical page address (chunkId in YAFFS2) and the physical address space is indexed by physical page address in the flash memory. This mapping can be either one-to-one mapping or one-to-many mapping. The former is used under the block device emulation and the latter is used mainly under the native flash memory file system. However, the YAFFS2 flash memory file system uses one-to-one mapping. A page in the flash memory contains a user data area and a spare area, where the user data area is for the storage of user data in a logical page and the spare area is for ECC, the corresponding logical page address, and other house-keeping data. When flash memory systems (flash memory file systems or block device emulations) are mounted, these mappings are constructed in the system memory to provide efficient file service. If these mappings are not constructed in the system memory when mounting the flash memory file system, then all pages in the flash memory must be scanned whenever accessing data. The log scanning proceduere conducted by a native file system will become a serious issue in the near future because flash memory's size is becoming larger. If the flash memory's size is becoming larger (2GB, 3GB,...), then the existing flash memory file systems (YAFFS2 and JFFS2) will be impractical. To reduce the initialization time of the log-based flash memory file systems, [5]'s and [6]'s method were proposed. [5]'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 for the flash memory management and the updated areas are not addressed easily. Snapshotting a file system image when unmounting, elongates the shutdown time regardless of unmounting improperly. To resolve [5]'s problems, [6]'s method was proposed. [6]'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 this paper, we propose our method to reduce the log-based flash memory file system's initialization time and recovery time and to provide efficient file system service (high performance and small system memory usage) for the multimedia files.

## 3 The Proposed Method for Log-Based Flash Memory File Systems

To reduce the log-based flash memory file system initialization time and the recovery time, our proposed method introduces the startup area. The startup

area is allocated separately from the data area (YAFFS2) and it is managed by SyncManager. The startup area's size must be determined according to usage of the embedded systems using this method. the startup area's size is determined in proportion to the number (not capacity) of the files stored in Flash memory. For example, the startup area need small size relatively for the embedded systems using large-size files (mp3, movies). [Fig. 3] shows the overview of our proposed method. We implemented the prototype based on YAFFS2 but our proposed method can be applied in other log-based flash memory file systems. To track the changes of the file system image, [6] records the changes of I/O operations in a log-segment but the number of log-segments increase as the I/O operations increase. It increases the crash recovery time. [6]'s method focused on I/O operations but our propsed 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 filter functions are inserted in YAFFS2's low-level I/O functions. 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 corressponding file. When a file is updated, the corresponding log page in the startup area is marked as DIRTY in the tag and the SynManager 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 syncronized with the YAFFS2 data-area, then the updated portions in the flash memory will be tracked by finding the DIRTY-marked log pages in the startup area, allowing fast crash recovery. The startup area is fixed in flash memory. 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. When SyncManager does garbage-collection in the startup area and the large portion of the blocks in the startup area is used, the cold blocks are selected as the garbage-collection's victims.

For high performance, our proposed method uses a different page addressing technique from that of YAFFS2. Each Tnode's entry in the YAFFS2 contains a group address, which is not a page address. In YAFFS2, the wanted page is

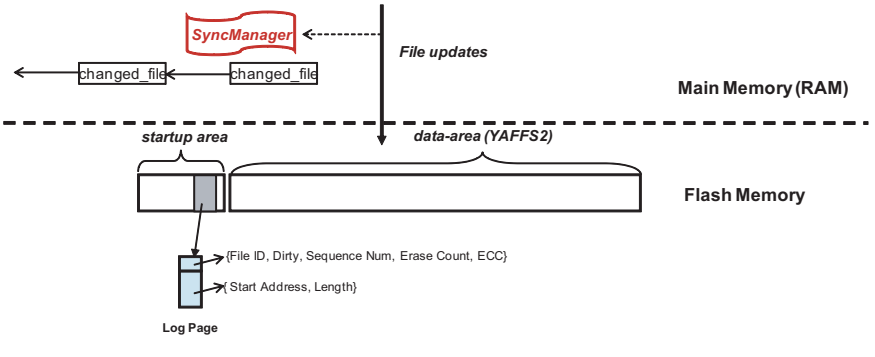


**Fig. 3.** Overview of the proposed method

found by searching pages in the group containg 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. However, this method consumes more system memory for directly addressing a physical page. Our proposed method uses the B-Tree technique for the data address mapping instead of YAFFS2's Tnode Tree, which allocates a Tnode with 16 entries for using a page but a node in our B-Tree has a entry and it can represent many pages if the pages is 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. A node in our B-Tree uses the more system memory than the YAFFS2's Tnode. But the B-Tree node represents the more information than the YAFFS2's Tnode and our B-Tree node is more effective in the multimedia data, which mainly is stored continuously 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 a file is unused for 5 minutes or the number of the used files exceed the guideline, 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. [Fig. 4] shows our proposed data mapping in the system memory.

## 4 Performance Evaluation

The experiments were conducted with YAFFS2 and our prototype, which was implemented based on YAFFS2. Our experimental setup is shown in [Fig. 5].

YAFFS2 runs with the "Erasure Check mode", which confirms whether each block in the flash memory has been erased on the mount time or not. Our prototype runs with the "Erasure Check mode" as well. If we do not use the "Erasure Check mode", then our proposed method's initializaton time will be shorter. [Fig. 6] and [Fig. 7] show the flash memory file system initialization time of our prototype and YAFFS2, using small-sized files (average size: 359.5 KB). [Fig. 6] and [Fig. 7] show that our prototype's initialization time is shorter than that of YAFFS2, since the startup area and eachblock's first page are scanned in our proposed method. [Fig. 8] and [Fig. 9] show the flash memory file system initialization time of our prototype and YAFFS2, using mp3 files, whose average file size is 5.9 MB. In [Fig. 8] and [Fig. 9], our prototype's initialization time is under 9 seconds but YAFFS2's initialization 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 mainly are stored, their size is large, and the number of the stored multimedia data is limited. Our proposed method records the changed files to track the changes of the file system image and writes the changed files' mappings into the startup area periodically.

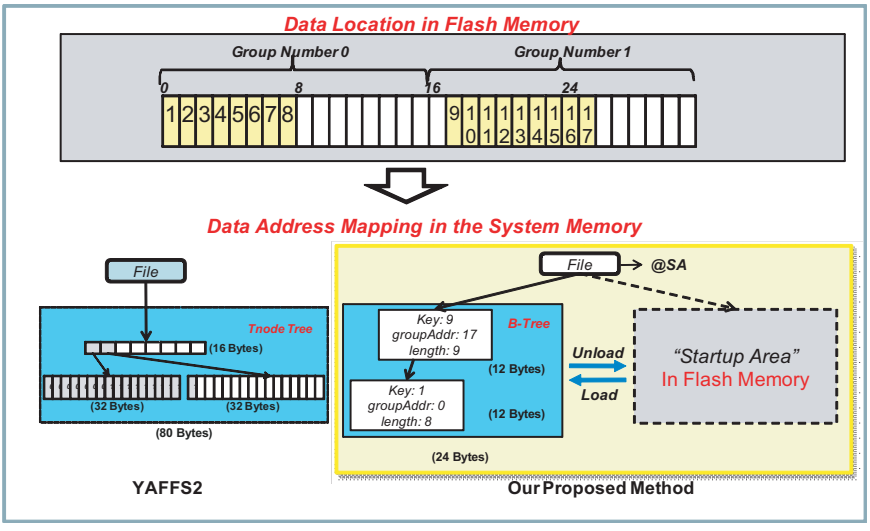


**Fig. 4.** Our Memory Management

| Item | Description |
|---|---|
| Processor | Intel Bulverde PXA270 (520Mhz) |
| SDRAM/Flash | Samsung SDRAM 64MB/Intel strata flash 32MB |
| Ethernet | CS8900A 10 Base T |
| USB | USB Host 1.1 & Slave 1.1 |
| NAND6EA | 1GB NAND Flash Memory x 6 |

**Fig. 5.** Experimental Setup

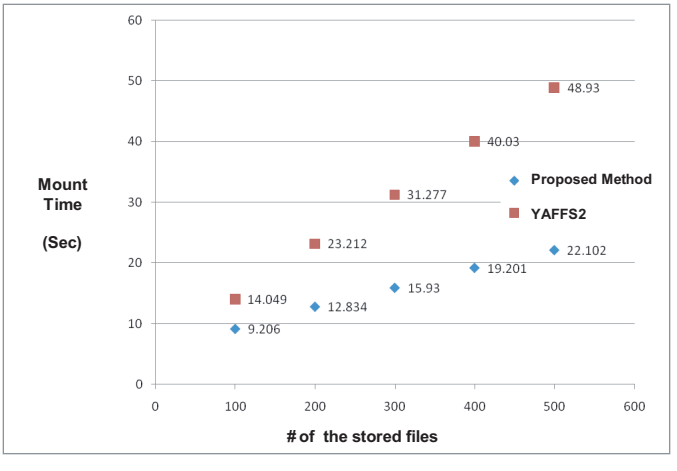


**Fig. 6.** Mount Time vs. Number of the stored files (average size of the stored files: 359.5KB)

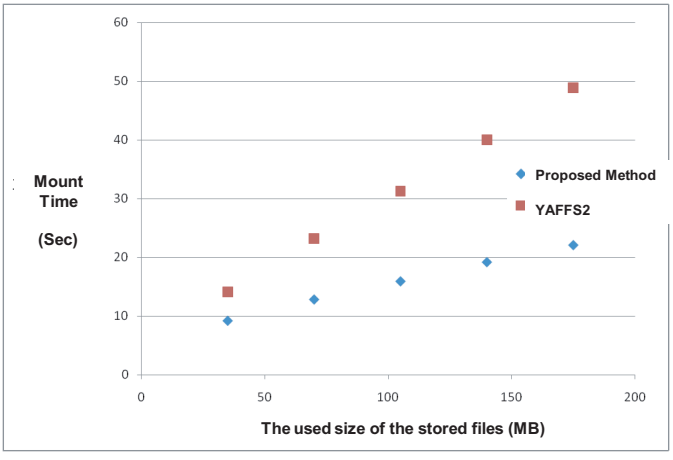


**Fig. 7.** Mount Time vs. The used size of the stored files (average size of the stored files: 359.5KB)

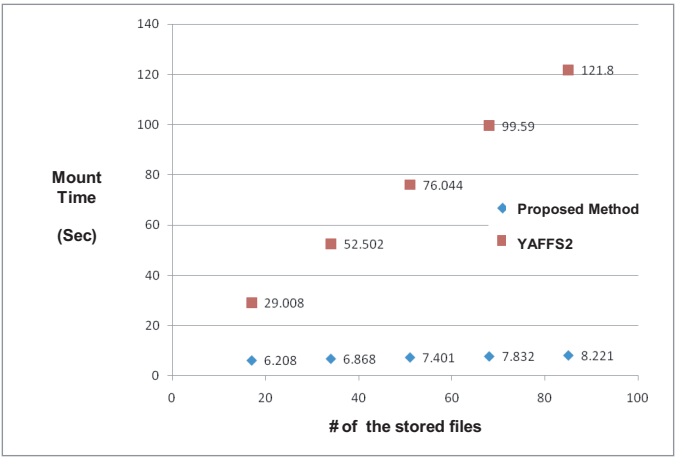


**Fig. 8.** Mount Time vs. Number of the stored files (average size of the stored files: 5.9 MB)

Therefore crash recovery time is very short because the updated portions in the flash memory can be addressed exactly.

To compare the performance of YAFFS2 and our prototype, we used iozone benchmark tool. [Fig. 10] showes that our proposed method outperforms YAFFS2, since our method uses the direct page mapping. When playing a movie encoded in MPEG4, 640x480, 24bits Per Pixel, 20fps by using a mplayer without a frame drop, our prototype is average 120.03 seconds faster than YAFFS2 as well. To measure the random seek performance in the large-sized multimedia

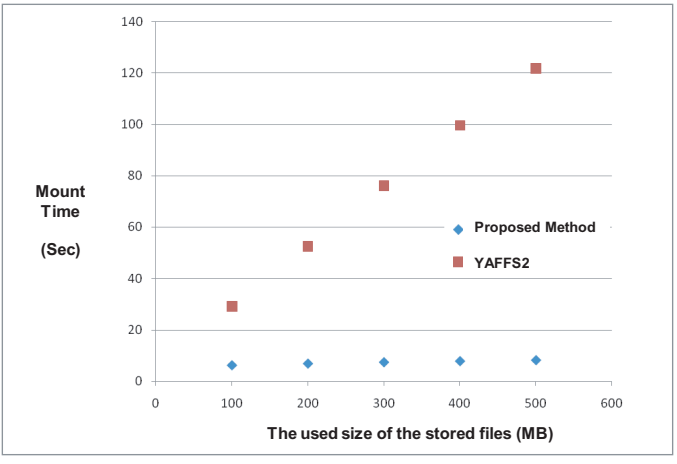


**Fig. 9.** Mount Time vs. The used size of the stored files (average size of the stored files: 5.9 MB)

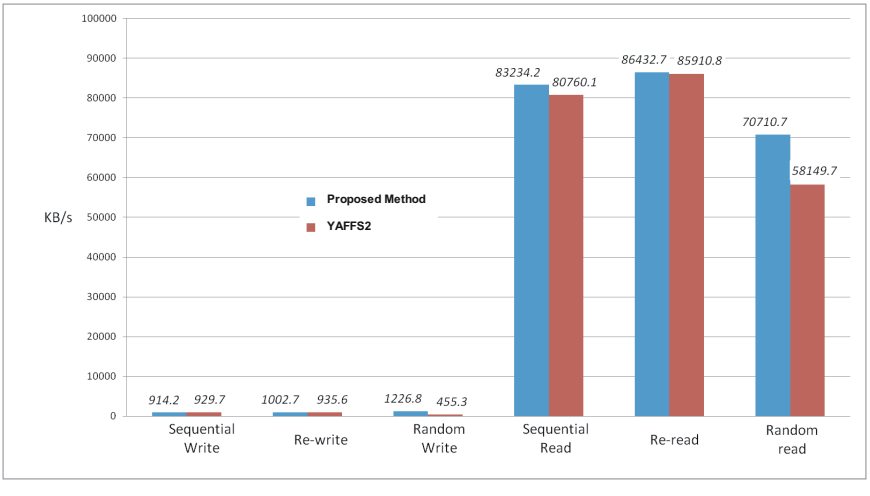


**Fig. 10.** Performance Test(iozone: transfer size - 4KB)

file, our test used a 321MB sized multimedia file and read 1 byte data at the the relative positions (0%, 90%, 20%, 70%, 40%, 50%, 60%, 30%, 80%, 10%, 100%) from the start of the file. When completing this test, YAFFS2 takes 0.211 seconds while our propopsed method takes 0.112 seconds when the data address mapping is unloaded and 0.012 seconds when the data address mapping is loaded previously. When the data address mapping is unloaded, the overhead is introduced to load the data address mapping.

## 5 Conclusion

We proposed a method for fast initialization and memory management for the log-based flash memory file systems. This method introduces the start up area allocated separately from the existing file system data area. The startup area is managed by SyncManager and contains log-pages. A log page has the corresponding file in the existing file system (YAFFS2) and the file's mapping data in the flash memory. When a system crash occurs, the unsyncronized files are detected easily by confirming the starup area. So the file system initialization and crash recovery are fast. The mapping address in a log page is a physical page address in the flash memory, hence the proposed method outperforms YAFFS2, which has group address mapping. Our proposed method may not use the system memory to maintain the files' address mappings because when it is needed, it can be loaded from the startup area.

## Acknowledgment

This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment)(IITA-2006-C1090-0603-0045).

## References

1. Aleph One Company, " Yet Another Flash File System".
2. D. Woodhouse, Redhat, Inc., "JFFS: The Journalling Flash File System".
3. Compact Flash Association, " CompactFlash 1.4 Specification," 1998
4. Intel Corporation, "Understanding the Flash Translation Layer (FTL) Specification".
5. Keun Soo Yim, Jihong Kim, and Kern Koh, "A Fast Start-Up Technique for Flash Memory Based Computing Systems," Proceedings of the 2005 ACM Symposium on Applied Computing
6. Chin-Hsien Wu, Tei-Wei Kuo, and Li-Pin Chang, " Efficient Initialization and Crash Recovery for Log-based File Systems over Flash Memory," Proceedings of the 2005 ACM Symposium on Applied Computing
7. M. Rosenblum, and J.K. Ousterhout, " The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems 10(1) (1992)