JAM: Justifiable Allocation of Memory with Efficient Mounting and Fast Crash Recovery for NAND Flash Memory File Systems

Sanam-Shahla Rizvi and Tae-Sun Chung
School of Information and Computer Engineering, Ajou University, Korea

Abstract: *Flash memory is small size, lightweight, shock-resistant, non-volatile, and consumes little power. Flash memory therefore shows promise for use in storage devices for consumer electronics, mobile computers and embedded systems. Even though, flash memory has many attractive features but issues on performance and data integrity are becoming more critical to address by researchers. First, the rapidly increasing capacity of flash memory imposes long mount time delay for normal start-up and in case of crash recovery. Second, large main memory requirement for keeping file system mapping data structure becoming significant issue with growth in size of flash memory. In this paper, we discuss related problems in detail, and propose novel mechanism for high performance and system reliability by effective metadata management, efficient mounting, fast crash recovery, and reduced RAM footprints for log structured NAND flash memory based file systems, called justifiable allocation of memory. The trace driven simulation results show the significantly improved performance for mounting and crash recovery time with reduced main memory space required by our proposed justifiable allocation of memory scheme compared to well-known JFFS2 and YAFFS2 flash file systems.*

Keywords: *Consumer electronics, embedded systems, memory management, system crash recovery, system reliability, and system performance.*

Received December 18, 2008; accepted July 26, 2009

1. Introduction

Flash memory is a non-volatile solid state memory, which has many attractive features such as small size, fast access speed, shock resistance, high reliability, and light weight. Because of these attractive features, and decreasing price and increasing capacity, flash memory is becoming ideal storage media for consumer electronics, embedded systems, and wireless devices. Furthermore, its density and I/O performance have improved to a level at which it can be used as an auxiliary storage for mobile computing devices, such as PDA and laptop computers.

NAND flash memory is partitioned into equal size of erase units called blocks and each block is composed of fixed number of read/write units called pages. Every page has two sections, data area and spare area. Spare area, as shown in Figure 1, stores metadata like Logical Block Number (LBN), Logical Page Number (LPN), Erase Count Number (ECN), Error Correction Code (ECC), cleaning flag for indicating garbage collection process in block, used/free flag to show page is used or still free, and information of being valid/obsolete about data in data area. The size of page and block differs by products.

Flash blocks are logically divided by the type of data stored in them. Figure 2 shows the complete logical structure of memory blocks. There are five types of blocks. Data blocks hold ordinary user data and log blocks store update writes to data blocks. Dirty blocks hold obsolete data and ready to be erased. Free blocks are used to be assigned for new data. Finally, map blocks store system metadata about all above types of blocks. Blocks holding user data, as data blocks and log blocks, split in hot and cold blocks by the frequency of data modification. The hot blocks hold data with frequently updating nature and the cold blocks hold data with infrequently updating or of read only nature.

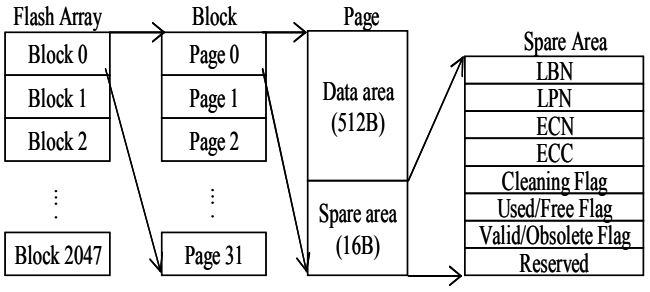

Figure 1. Flash memory (32MB) architecture.

At the time of normal booting, system fetches the metadata from map blocks and builds the address mapping structure in main memory for fast access of flash storage media. The main memory size in mobile devices does not follow the trend of increasing size of

flash memory. Thus keeping large size of mapping data structure in RAM is becoming big challenge.

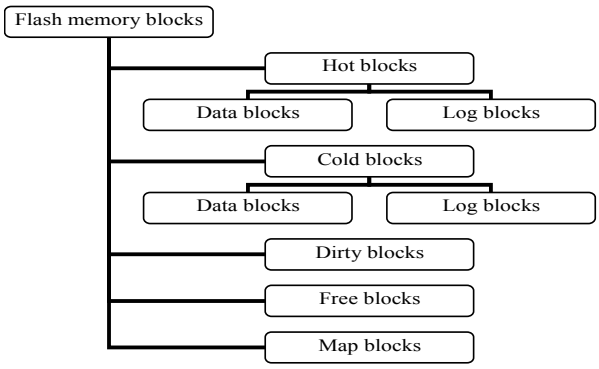

Figure 2. Logical structure of flash memory blocks.

Metadata from main memory stores back to map blocks whenever it changes. Many of embedded systems rely on battery backed power, and sudden power-off results in system crash. It flushes all data from volatile main memory, and leaves the metadata unreliable in flash map blocks as well. After power-recovery, previous schemes as [4, 8] regain reliable state by scanning entire media. Such scan-to-initialize approach is neither time nor power efficient, and also not practical for large size flash memories.

In this paper, we propose novel idea for efficient mounting and fast crash recovery for high performance and system reliability for log structured NAND flash memory based file systems, called Justifiable Allocation of Memory (JAM). The objectives of our present research are to implement an effective metadata management strategy to minimize the required time for mapping table construction, both for normal start-up and in case of crash recovery, and to achieve small main memory footprints.

The remainder of this paper is organized as follows. We review the existing work in section 2. Proposed mounting and crash recovery schemes are presented in section 3, and evaluation results are discussed in section 4. Finally, paper concludes in section 5.

2. Related Work

JFFS2 [8] and YAFFS2 [4] are well-known general-purpose log-structured flash file systems. JFFS2 developed for embedded Linux and YAFFS2 is designed specifically for NAND flash memories for embedded devices and has been using in products running both Linux and Windows consumer electronics. Both JFFS2 and YAFFS2 are freely available under the GNU Public License (GPL).

Both file systems store data in flash array sequentially in incremental order irrespective of data nature. JFFS2 stores data and its corresponding metadata together in data area when writing data to flash memory. At mounting time, it scans the entire flash array to construct the mapping structure in main memory.

Unlike JFFS2, YAFFS2 considers the characteristics of flash memory as read/write units and their corresponding spare regions. It stores the metadata in spare areas and at the mounting time it scans only the spare regions for collecting mapping information. Therefore, it outperforms JFFS2 with respect to mount time and amount of main memory consumption. Both file systems supporting lengthy and time consuming scan-to-initialize technique to construct mapping data structure at every mounting time. Therefore, time required for crash recovery is same as normal booting time. The initialization time, I/O computation time and main memory usage by both file systems significantly increases with stored data size and with growth of flash memory capacity.

Other previously proposed, mounting and crash recovery schemes for flash memory, as [1, 5, 9, 10] are not effective for small size; time, energy and main memory constrained embedded systems and wireless devices. As, such schemes require large space in main memory, heavy I/O operations, and maintain extra space in flash for supporting their techniques.

3. Proposed JAM Scheme

3.1. Proposed Mounting Technique

In this section, we achieve efficient and fast mounting with reduced main memory footprints by our proposed monitoring module, as shown in system architecture in Figure 3.

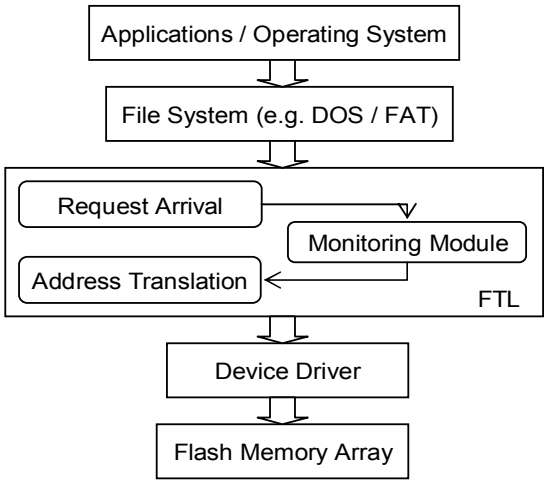

Figure 3. Proposed system architecture.

The proposed monitoring module is responsible for measuring the memory accessibility ratio on the level of granularity of blocks. Module applies on every data block, both hot and cold nature, and monitors the request arrival rate by function as equation 1.

$$\lambda = \begin{cases} 0 : ni \ (I) < TH \\ 1 : ni \ (I) \geq TH \end{cases} \quad (1)$$

If the number of request arrivals *"ni"* crosses the predefined threshold of workload *"TH"* in particular time interval *"I"* than the metadata of block is marked feasible to access in main memory from map blocks at the next mounting time else block will not be fetched. The blocks those are fetched in RAM later on demand will also be under monitoring to decide if some block becomes accessible for next mounting. We keep the final block mo nitoring status "λ" in RAM and also store this information in spare areas of map blocks to identify the frequently accessed blocks on next time mounting.

Based on monitoring module results, we achieve reduced main memory footprints by following two techniques.

- Initial RAM footprints: mounting only frequently accessed metadata initially in main memory on system start-up.
  We are particularly motivated to consider the monitoring module estimation status for cold blocks and read only blocks, because those blocks are observed in majority, so mounting such blocks without monitoring increases the main memory consumption without sufficient use. Therefore, infrequently accessed blocks are fetched on demand by using dynamic memory allocation. The dynamic memory allocation is used not to reserve the constant space in main memory. It provides opportunity to allocate RAM space dynamically on run time when it is required. Therefore, the infrequently accessed metadata can be called from map blocks when some request arrives. This approach may take some times frequent read operations to map blocks but real time workload proves that majority of data is accessed once in a while. This approach highly reduces the consumption of RAM space.
- RAM footprints reduction on run time: Discarding the mapping information of blocks those have not been accessed for long time.
  The status of monitoring module is saved consecutively in RAM with every block to show its accessibility ratio. Therefore, we extend our idea to reduce the required RAM space for keeping mapping information by removing the mapping structures of blocks those have not been accessed during previous monitoring interval. This idea not only preserves the main memory space but also reduces the write operations required to update the metadata on map blocks. This strategy proves effective for the devices those have very limited RAM resources like wireless sensor nodes. Therefore, this approach may cause some times frequent read operations to map block.

3.2. Proposed Crash Recovery Technique

In this section, we propose a technique to minimize the data loss due to unexpected power-off, and we offer fast and efficient crash recovery by "intelligent" procedural computation in main memory based on latest available snapshot of file system. In case of sudden power-off, crash may leave the system in one of three states. These states are related to user data, metadata, and erase operation. We discuss all possible flash states with different crash scenarios, and gradually define the proposed recovery approaches.

3.2.1. User-Data Crash Recovery

Crash scenario: in case of sudden power-off, crash can happen during writing user data either on data blocks or on log blocks. The write operation usually has following steps:

a. Write data on exact offset page in data block, or on available page in log block.
b. Write corresponding LPN in spare area.
c. Mark new data page as valid.
d. Mark old data page with same logical address, if any, as invalid.

Flash stands in one of following states, when unexpected power-off occurs:

- During writing data in data area, as shown in Figure 4(i).
- After writing data in data area, but before writing LPN in spare area, as shown in Figure 4(ii).
- After writing data and LPN, but before marking page as valid in spare area, as shown in Figure 4(iii).
- After writing data, LPN and marking new page as valid, as shown in Figure 4(iv), but before marking old page, if any, as invalid in spare area.

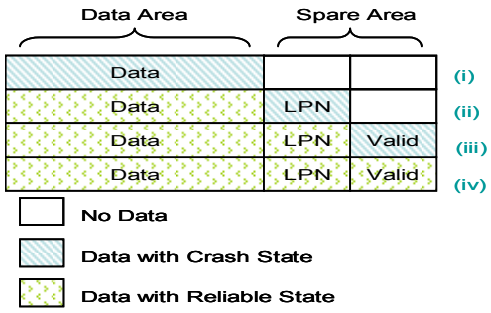

Figure 4. Crash states for user data.

Recovery: while booting after power recovery, the victim page is identified by ECC flag. ECC flag is used against error during write operation in data area or spare area. Recovery is provided to above discussed crash states as follows:

a. In case of first and second state, as the data would be unrecognizable due to LPN not written in spare area, so the new page is marked as invalid and its obsolete flag set high to mark it ready for erasure.
b. For third state recovery, as data is properly written in data area and it is recognizable by its LPN in

spare area, so new page is considered as valid to minimize the data loss and to provide appropriate reliability.

c. Same as in fourth state but then the ambiguity between old and new page is resolved by marking old page with same logical address as invalid.

3.2.2. Metadata Crash Recovery

Crash scenario: in log based flash file systems under block device emulation, memory mapping information in main memory and in map blocks changes with the allocation of new data block or log block as in [2, 3] by following conditions.

- Block allocation for completely new data: system allocates new data block when completely new data first time arrives to be stored in memory. New log block assigns to store updates to data in data blocks.
- Split operation: when assigning new log block crosses the predefined limit of log blocks, system triggers the garbage collection on log blocks. System selects the victim log block according to its victim selection policy. Reclamation applies by Split operation, as in [6], where valid data from old log block copies to new allocated log block, then former block marks obsolete and moves to dirty blocks pool for erasure. Then the further updates to data blocks forward to new log block.
- Merge operation: when system crosses the predetermined threshold as maximum allowable utilization of media, it triggers the garbage collection on data blocks. System selects the victim data block according to its victim selection policy. The victim data blocks is reclaimed by Merge operation as in [3], where valid data from old data block and its corresponding log blocks copies to new allocated free block and marks former data and log blocks as obsolete and moves to dirty blocks pool for erasure. Then the new allocated block responses as data block for future transactions.

New block allocation involves four updates as free block becomes new data or log block, and old data or log block becomes dirty block, and that dirty block turn into part of free blocks after erasure. It requires updating metadata, as data blocks list, log blocks list, dirty blocks list, and free blocks list, according to condition.

The state of map blocks becomes unreliable if power loss occurs after new block allocation but before updating metadata on map blocks.

Recovery: while booting after power revival, the verification and recovery of the consistency of latest available snapshot in map blocks is obtained step by step as follows.

- Verify consistency of mapping structure.

a. Fetch latest available file system snapshot from map blocks in main memory by function as equation 2.

$$FetchFileStructure() \quad (2)$$

b. Extract PBNs of high and low ECN blocks from free blocks list, as PBN15 and PBN16, as shown in Figure 5.

Assumption: in this paper, we assume that flash blocks consist of four pages for simplicity of examples in figures.

c. Check the free flag status high of extracted blocks in their block header by function as equation 3.

$$IsFree\ (PBN_{LowECN}, PBN_{HighECN}) \quad (3)$$

d. If both blocks with high and low ECNs are still free than the latest available file system snapshot is consistent.

e. Skip further checking and use same information for future transactions.

f. Else, if the latest available mapping structure is not consistent than verify the lastly allocated and obsolete blocks intelligently based on old snapshots, as described in following steps.

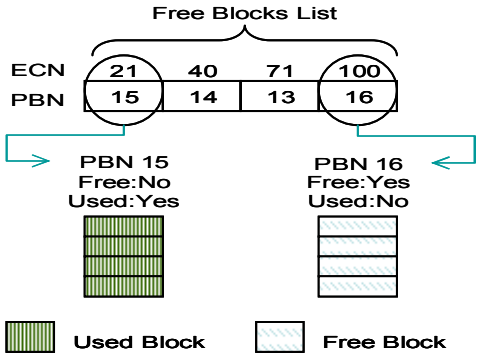

Figure 5. Lastly allocated block recovery.

- Verify lastly allocated block

a. Extract PBNs of high and low ECN blocks from free blocks list, as PBN15 and PBN16, as shown in Figure 5.

b. Identify lastly allocated block by its used flag status high in its block header by function as equation 4 and mark it in corresponding hot or cold blocks pool.

$$IsUsed\ (PBN_{LowECN}, PBN_{HighECN}) \quad (4)$$

Assumption: in our present scheme, we assume that a new data or log block allots with high erase count for cold nature data and with low erase count for hot nature data for the sake of wear-levelling. Therefore, wear-levelling is a process to evenly distribute the erasure on all blocks to prolong the life of flash media against limited number of allowed erase operations.

In Figure 5, PBN15 is newly allocated block as per its used flag status high and it assigns to hot blocks pool by its low erase count.

c. Reorganize mapping information in main memory, as hot blocks pool increases and free blocks pool decreases, and update up-to-date file structure in map blocks by function as equation 5.

$$CommitFileStructure() \quad (5)$$

- Verify lastly obsolete blocks.

After designating the lastly allocated block in its corresponding block nature pool, check the consistency of dirty blocks pool by following steps.

a. Extract the LBN of lastly allocated block from its block header as LBN15 of PBN15, as shown in Figure 6.
b. Search the blocks with same LBN from metadata of data and log blocks lists as PBN1 and PBN5. This search leads to the blocks lastly obsolete due to Split or Merge operation.
c. Verify their status by their obsolete flags high in their block headers by function as equation 6, as shown in Figure 6, and mark them in dirty blocks pool.

$$IsObsolete\ (PBNa_{LBN}, PBNB_{LBN}) \quad \therefore \quad a \neq b \quad (6)$$

d. Reorganize mapping information in main memory and update up-to-date structure in map blocks by function as equation 5, as shown in Figure 6 old data block PBN1 and old log block PBN5 become part of dirty blocks pool and data and log blocks pool decreases by both obsolete blocks.

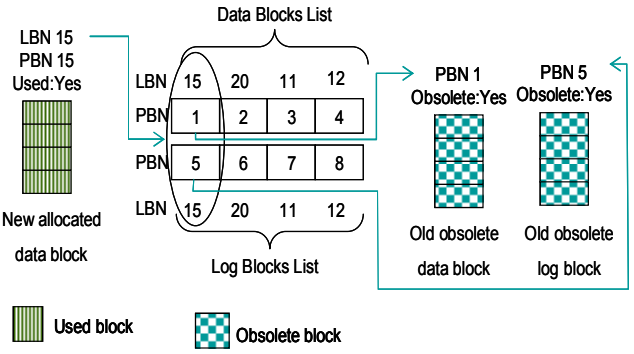

Figure 6. Lastly obsolete blocks recovery.

3.2.3. Erase Operation Crash Recovery

Crash Scenario: system keeps obsolete blocks in dirty blocks pool and erases in idle time of system or when system crosses the threshold of free blocks. System maintains the ECN of block in block header and on map blocks with corresponding LBN.
Assumption: in our present scheme, we assume that system always selects a victim block for erasure with low ECN from dirty blocks pool for the sake of wear-levelling.

After erasure of victim block, system rewrites increased ECN in block header and block is moved to free blocks pool. The process of erasure requires updating two entries in metadata to commit, as free blocks list increases and dirty blocks list decreases. System stands in one of the following states, when light-off occurs:

a. During erase process, so ECN of victim block is unrecoverable as block header is also erased along with volatile main memory.
b. After block erased properly, and it's increased ECN have been written in block header, but before updating meta-information in map blocks.

In case of unexpected power loss during erase process, causes the loss of ECN on block was being erased and from mapping information in main memory. It also results in unreliable metadata on map blocks. In first state of system crash, when power-off occurs during erase process, only dirty blocks list effected, but in second state, when block is properly erased but mapping information may not updated due to unexpected power-off, the metadata of both dirty blocks and free blocks becomes unreliable.
Recovery: while booting after power recovery, the ECN plus stable metadata achieves based on latest available snapshot in map blocks by performing following steps.

- Recover erase count number: to recover the reliable state of system crash when power-off occurs during erase process, and the victim block was not properly erased:

a. It needs to erase the victim block again, as shown in Figure 7(i).
b. Fetch the latest available file system snapshot from map blocks in main memory by function as equation 2.
c. The previous ECN of victim block is recovered by selecting the low ECN block available in dirty blocks pool; as shown in Figure 7(ii).

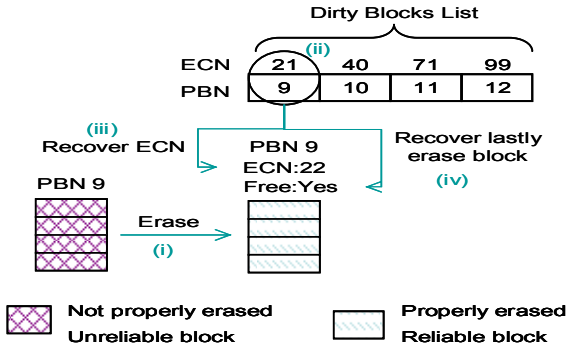

Figure 7. Erase operation recovery.

The victim block and the block with low ECN from dirty blocks pool both points out the same block

because the selection of block for erasure is based on low erase count.

d. Store the increased ECN in recently erased block header, as shown in Figure 7(iii), and move block to free blocks pool.
e. Reorganize the mapping information in main memory accordingly, as dirty blocks list decreases and free blocks list increases, and update in map blocks by function as equation 5.

- Recover mapping structure: to recover the sustainable state of system crash when power-off occurs after block is properly erased but before updating mapping information in map block:

a. Fetch the latest available file system snapshot from map blocks in main memory by function as equation 2.
b. Compare the low erase count block from dirty blocks list by its PBN to free blocks list.
c. If block is not available in free blocks list, confirm the block information by reading its free flag status high from block header by function as equation 3, as shown in Figure 7(iv).
d. Reorganize the mapping information in main memory accordingly, as dirty blocks list decreases and free blocks list increases, and update in map blocks by function as equation 5.

4. Performance Evaluation

4.1. Simulation Methodology

To evaluate the performance characteristics of JFFS2 [8], YAFFS2 [4], and our proposed JAM scheme, we developed simulator for each scheme and performed trace-driven simulations. We built a simulator with 32 megabytes of flash space that is divided into equal size of erase blocks. Each block size is 16 kilobytes and every block is composed of 32 pages as read/write units. Every page data area size is 512 bytes with 16 bytes spare area. We use 15µs for a page read, 200µs for a page write, and 2ms for a block erase from [7] product.

We use five data traces in this experiment, as shown in Table 1. These traces have been obtained from the author of [3]. Traces *A*, *B* and *C* are generated from digital cameras and thus contain both small random inputs and large sequential inputs. Traces *D* and *E* contains many small random inputs than large sequential inputs.

For each given trace, system compares our proposed scheme JAM with JAFFS2 and YAFFS2 for required main memory space and time for normal mounting and in case of crash recovery. We believe that these traces are complex enough to show the characteristics of our proposed scheme for more general flash memory based systems.

The total elapsed time is calculated by equation 7 for effective comparison between all schemes. Time required for read in unit of page from flash memory to data register is calculated by equation 8. Time required for read in unit of byte from data register to main memory is calculated by equation 9. Time required for computation in main memory for building mapping structure is calculated by equation 10. Usually main memory time is very small to neglect for small size operations. Time required for write back, if any, the mapping structure from main memory to flash media is calculated by equation 11.

Table 1. Simulation traces.

Traces	Workload Description	Number of Inputs
A	Digital camera (A company)	23518
B	Digital camera (B company)	74687
C	Digital camera (C company)	139152
D	Linux O/S	56700
E	Symbian O/S	32392

$$Total\ time = TR_{FR} + TR_{RR} + Ta + TW_{RF} \quad (7)$$

$$TR_{FR} = read\ count\ (page) x\ read\ time) \quad (8)$$

$$TR_{rR} = read\ count\ (byte) x\ read\ time) \quad (9)$$

$$Ta = Time\ for\ Computation\ in\ RAM \quad (10)$$

$$TW_{RF} = (writecount(page) x\ writetime) \quad (11)$$

4.2. Experimental Results

Figures 8 and 9 present the results of amount of space consumed in RAM in unit of kilobytes while mounting for normal start-up and in case of crash recovery, respectively. Results show that our proposed scheme JAM highly outperforms to both native approaches, as JFFS2 and YAFFS2. Both schemes store logical/physical mapping information on page granularity level and the time and space required for address translation structures highly depends on stored data size. Our JAM scheme examines the request arrival ratio by smaller threshold on every logical block by monitoring module to reduce the number of blocks required to fetch in RAM on the time of mounting. Threshold *"TH=3"* shows that the metadata of blocks from map blocks are fetched, those were accessed atleast three times in last monitoring time interval. Dynamic block allocation considerably reduces the RAM footprints. We can recognize the efficiency of our metadata management based on data nature, which shows the demand to handle data by their accessibility ratio. In real workload, usually data access patterns changes with time, in that case monitoring module is best choice to identify the blocks those are not accessed for long period of time to save the expansive RAM space and I/O computation while fetching the metadata in main memory. Our proposed JAM scheme ensures recovery highly based on metadata and intelligent procedures. Therefore, for recovery, JAM consumes space in main memory to keep only system data and few more

pages, if required, from flash media. We offer less RAM space consumption overall 99.4% and 98.8% while mounting and 99% and 98% while crash recovery compared to JFFS2 and YAFFS2, respectively.

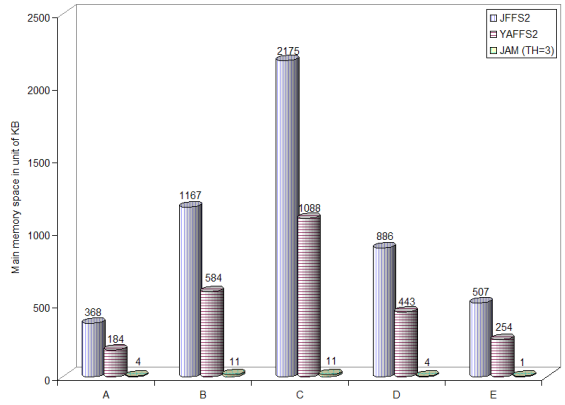

Figure 8. Space consumed in RAM (KB) for system mounting.

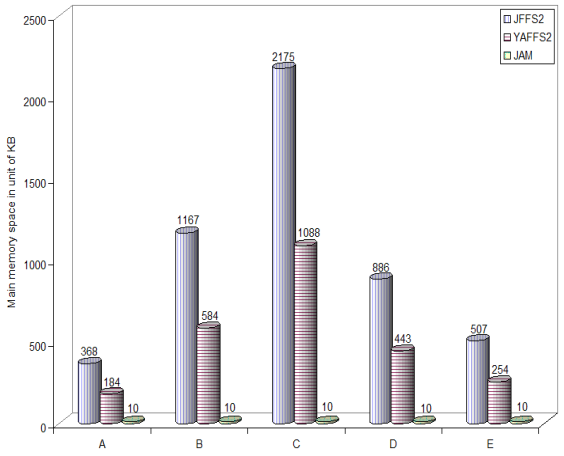

Figure 9. Space consumed in RAM (KB) for system recovery.

Figures 10 and 11 present the results of required time in unit of millisecond while mounting for normal start-up and in case of crash recovery. Our proposed scheme JAM outperforms both schemes as JFFS2 and YAFFS2. The reason of high performance is that we fetch mapping information from dedicated map blocks while booting rather than scanning whole flash media. But JFFS2 scans entire flash media to extract mapping information as both data and metadata are saved together in data section, and YAFFS2 scans the spare areas of all used data sectors on whole flash media to reconstruct the file system on every booting time. Therefore, due to same scan-to-initialize way, both schemes give same results for start-up in both cases as for normal start-up and in case of crash recovery, and highly decrease system performance. Even though, YAFFS2 gives better results compared to JFFS2 but they both suffers compared to our proposed scheme JAM. To efficiently handle the problem of data reliability, JAM scheme uses the already available old metadata in map blocks and reconstructs the file system by effective computations in RAM, intelligently, as discussed in detail in Section 3.2. Growth in data storage increases the mounting and recovery time for JFFS2 and YAFFS2 but our JAM scheme offers constant recovery time for all situations. We offer faster mounting performance overall 99.9%, and prove faster recovery performance overall 99.7%, compared to both schemes as JFFS2 and YAFFS2.

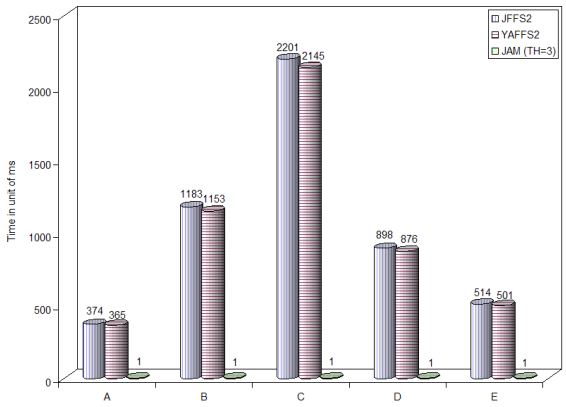

Figure 10. Time (ms) for system mounting.

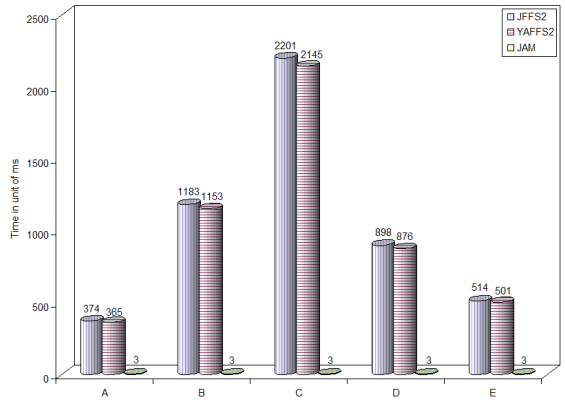

Figure 11. Time (ms) for system recovery.

5. Conclusions

This research proposes the novel JAM system software scheme to ensure the high performance and system reliability by innovative metadata management, efficient mounting, fast crash recovery, and reduced main memory footprints for NAND flash memory file systems. Our present research relates to minimize the required time and main memory data structure for mapping table construction, both for normal start-up and in case of crash recovery. Monitoring module used to ensure the effective metadata management. Reduced main memory consumption achieved by efficient mounting based on data accessibility ratio. Data consistency issues discussed in detail and provided effective fast crash recovery. The file system reliability improved mostly on behalf of metadata rather than scanning whole flash media. Through comprehensive evaluations, we proved that JAM has outstanding performance compared to JFFS2 and YAFFS2.

We demonstrated that our proposed JAM scheme is more time, energy and space efficient. Therefore, it proves an ideal technique for small size; time, energy and run time memory constrained devices like wireless nodes, ubiquitous devices, pervasive applications and other embedded systems and consumer electronics.

Acknowledgements

We wish to thank Mr. Muneer Ali Shah Rizvi, professor and dean, Greenwich University, Karachi Pakistan, and Mr. Vaqar Ayub Khamisani, Customer & Partner Experience Lead, Microsoft Ltd., UK, and Mr. S. M. Saif Shams, PhD Candidate, Simula School of Research and Innovation, Norway for their valuable time for reviewing the whole manuscript and responding with their helpful comments.

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract (UD060048AD).

References

[1] Chung S., Lee M., Ryu Y., and Lee K., "PORCE: An Efficient Power off Recovery Scheme for Flash Memory," *Computer Journal of Systems Architecture: Embedded Systems Design*, vol. 54, no. 10, pp. 935-943, 2008.

[2] Kim J., Kim M., Noh H., Min L., and Cho Y., "A Space-Efficient Flash Translation Layer for Compact Flash Systems," *Computer Journal of IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366-375, 2002.

[3] Lee W., Park J., Chung S., Lee H., Park S., and Song J., "A Log Buffer Based Flash Translation Layer Using Fully Associative Sector Translation," *Computer Journal of ACM Transactions on Embedded Computing Systems*, vol. 6, no. 3, pp. 68-83, 2007.

[4] One A., "Yet Another Flash Filing System," *Electronic Document*, http://www.aleph1.co.uk /yaffs/index.html, Cambridge, UK.

[5] Ryu J. and Park C., "Fast Initialization and Memory Management Techniques for Log-Based Flash Memory File Systems," *in Proceedings of the International Conference on Embedded Software and Systems*, Korea, pp. 219-228, 2007.

[6] Ryu Y., Chung S., and Lee M., "A Space-Efficient Flash Memory Software for Mobile Devices," *in Proceedings of International Conference on Computational Science and its Applications*, USA, pp. 72-78, 2005.

[7] Samsung Electronics, "NAND Flash Memory," K9F5608X0D Data Book, 2009.

[8] Woodhouse D., "JFFS: the Journaling Flash File System," *Ottawa Linux Symposium,* http://sources.redhat.com/jffs2/jffs2.pdf, 2001.

[9] Wu H., Kuo W., and Chang P., "Efficient Initialization and Crash Recovery for Log based File Systems over Flash Memory," *in Proceedings of the ACM Symposium on Applied Computing*, France, pp. 896-900, 2006.

[10] Yim S., Kim J., and Koh K., "A Fast Start-Up Technique for Flash Memory Based Computing Systems," *in Proceedings of the ACM Symposium on Applied Computing*, USA, pp. 843-849, 2005.

Sanam-Shahla Rizvi received the BCS degree in computer science from Shah Abdul Latif University, Khairpur, Pakistan, in 2003, and the MCS degree in computer science from KASBIT University, Karachi, Pakistan, in 2004, and MS degree in computer science from Mohammad Ali Jinnah University, Karachi, Pakistan, in 2006. She is currently candidate of PhD at School of Information and Computer Engineering at Ajou University, Korea.

Tae-Sun Chung received the BS degree in computer science from KAIST, in February 1995, and the MS and PhD degree in computer science from Seoul National University, in February 1997 and August 2002, respectively. He is currently an associate professor at School of Information and Computer Engineering at Ajou University. His current research interests include flash memory storages, XML databases, and database systems.