AMI: An Advanced Endurance Management Technique for Flash Memory Storage 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, nonvolatile, and consumes little power. Flash memory therefore shows promise for use in storage devices for consumer electronics, mobile computers, wireless devices and embedded systems. However, flash memory cannot be overwritten unless erased in advance. Erase operations are slow that usually decrease system performance and consume power. The number of erase cycles is also limited, and a single worn-out block affects the usefulness of entire flash memory device. Therefore, for power conservation, better system performance and longer flash memory lifetime, system support for erasure management is necessary. In this paper, we propose a novel idea of system software for garbage collection and wear-leveling called Allocation of Memory Intellectually for NAND flash memories. Proposed scheme classifies data blocks intellectually according to their write access frequencies and improves the space utilization by allocating separate limited number of log blocks to both natures, hot and cold, of data blocks with proposed new system architecture. Our proposed cleaning scheme achieves a block to erase with optimal number of space utilization and minimum overhead of data migration. A hybrid wear-leveling approach is also proposed to evenly wear-down flash memory. Proposed scheme enhances the system life time by managing the blocks according to their degree of worn. We compared our proposed idea with two previous schemes. Our proposed idea improved system performance 95% for garbage collection and 36% for wear-leveling. The evaluation results prove that our proposed scheme, AMI, outperforms both previous schemes particularly with efficient flash bandwidth utilization and attempted erase operations.*

Keywords: *Consumer electronics, embedded systems, data organization, endurance management, memory management, and system performance.*

Received November 5, 2008; accepted November 5, 2008

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 increasingly important storage medium for various computing systems, such as 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 media for mobile computing devices, such as PDA and laptop computers.

Flash memory is partitioned into equal size of erase units called blocks and each block has a fixed number of read/write units called pages. Flash memory has three kinds of operations: page read, page write, and block erase. The size of page and block differs by products.

Even though, flash memory has many attractive features, but its special hardware characteristics impose design challenges on storage systems. However, it has two main drawbacks.

First Drawback: an inefficiency of in-place-update operation. When we update data, we can not write new data directly at same address due to physical erase-before-write characteristics of flash memory. Therefore, updating even one byte data requires one slow erase operation before the new data can be rewritten.

To address the problem of in-place-update operation, the system software called Flash Translation Layer (FTL) introduced, as [6, 7, 9, 11]. FTL uses non-in-place-update mechanism, originally based on log-structured file system [13]. FTL avoids having to erase on every data update by using the logical to physical address mapping table, maintained in RAM. Under this mechanism, FTL remaps each update request to different empty location and then the mapping table is updated according to newly changed logical/physical addresses. This protects one block from being erased per overwrite. The obsolete data flagged as garbage, which a software cleaning process later reclaims, as [3, 5, 8]. This process is called garbage collection.

Second Drawback: the number of erase operations allowed to each block is limited (e.g., 10,000 to 1,000,000 times), and the single worn-out block affects the usefulness of the entire flash memory device. Therefore, data must be written evenly to all blocks. This operation is named as wear-leveling, as [1, 2, 4]. These drawbacks become hurdle for developing the reliable flash memory based systems.

Although researchers have proposed admirable garbage collection and wear-leveling policies, but they compromise for saving cleaning cost on flash memory bandwidth utilization and expansive main memory requirement as [2, 12, 16]. Therefore, still there is gape for a technique that could combine efficient erasure management with optimum usage of flash media and reduced cleaning cost. In this paper, we propose novel system software for garbage collection and wear-leveling, called Allocation of Memory Intellectually (AMI) for NAND flash memories. As the scheme name shows, we intellectually implement our idea based on diverse nature of data. We classify data according to its write access frequencies in hot and cold data nature, where hot data is frequently updating data, and cold data is infrequently updating and read only data. We store data in separate blocks by managing blocks according to their degree of worn. To provide free space for new data, our proposed cleaning scheme selects victim block for garbage collection based on ratio of block utilization and data migration cost. To achieve the evenly wear-down media, we propose hybrid wear-leveling that is combination of both types dynamic and static of wear-leveling approaches. Our objective is to improve the endurance management of flash memory with efficient data organization to enhance the overall system performance. We compare our proposed scheme AMI with well-known greedy policy [16] for garbage collection, and with dual-pool algorithm [2] for wear-leveling. To evaluate all three schemes, we developed a simulator and performed trace driven simulations. Our scheme shows improved performance compare to both previous schemes. For our proposed scheme, AMI, we found very surprising and encouraging results concerned endurance management and significantly very low erase operations attempted with amazingly evenly wear-down the flash device.

The remainder of this paper is organized as follows. We review existing works on garbage collection and wear-leveling in section 2. In section 3, we propose our endurance management technique as cleaning scheme and hybrid wear-leveling scheme for flash memory. We evaluate the performance of the proposed policy and previous schemes in section 4. Finally, we conclude this paper in section 5.

2. Related Work

A number of techniques have been previously proposed to improve the system performance. In this section, we review greedy policy and dual-pool algorithm regarding their technical structures and weaknesses.

2.1. Greedy Policy

Wu *et al.,* [16] proposed the greedy policy for garbage collection where eNVy controller and separate battery backed SRAM is used for storage management. The in-place-update is provided by keeping array in expensive SRAM where they buffer modified pages. eNVy uses logical to physical page mapping on fine granularity level. That large size of mapping table is also part of SRAM storage.

Cleaning operation is triggered when there is no free space remaining to flush data from SRAM buffer to flash memory except one block is kept free to support cleaning process. For cleaning, greedy policy selects the block with largest amount of garbage with least cleaning work to recover large free space. Greedy policy tends to select a block in a first-in-first-out order. Therefore, it shows to perform well for uniform accesses but poor for high locality of references. Greedy policy supports not more than one segment cleaning at a time. To keep the average cleaning cost, it limits the live data of the total flash array at 80%. Data distribute in uniform fashion irrespective of data access patterns. Therefore, by frequent cleaning operations, all of the segments end up with mix distribution of hot and cold data, and the overall cleaning cost increases with time when there is data distribution with high localities. However, greedy policy does not consider wear-leveling.

2.2. Dual-Pool Algorithm

Li-Pin Chang [2] proposed Dual-Pool algorithm (DP) for wear-leveling. The algorithm is based on two principals. First, blocks are prevented from being overly worn by storing cold data. Second, blocks involved in wear-leveling are left alone until wear-leveling takes effect. Author implements his scheme by three basic methods. Dirty-Swap method is used for swapping data blocks among hot and cold blocks pool when user-configurable threshold is crossed. Hot-pool resizes and cold-Pool Resize methods are used when data access patterns become change.

Even though, this scheme claims attractive features regarding spatial localities but it has few intolerable drawbacks. First, DP uses smaller user-configurable threshold to pursue more even wear-leveling. Smaller threshold leads to frequent cold data migration, which results in repeated erase operations and additional I/O overhead. Therefore, ratio of erase cycles increases and system faces extra unnecessary cleaning overhead. Second, DP applies many checks to confirm wear-leveling condition after every write operation. Hence, small handheld devices which support limited computation, these checks after every write operation could cause system speed slow. Third, DP consumes extra RAM space to maintain effective-erasure-cycles with every block in RAM along with system required logical to physical mapping information and Erase Count Number (ECN) of every block. Therefore, devices like mobile phones and PDAs have limited RAM resources can suffer their performance because

of extra RAM consumption overhead. The user-data and meta-data storage management in flash memory is not exactly mentioned in DP scheme, except author mentions greedy policy for garbage collection.

3. AMI: Proposed Endurance Management Technique

3.1. System Architecture

The proposed system architecture is illustrated as Figure 1. The proposed garbage collection module consists of four components. Hot/Cold Data Identifier recognizes hot and cold data by their nature. The nature is accessibility ratio per logical block number for read/write operations, further discussed in detail in Section 3.2. This activity helps in assigning free blocks for data storage. Free Block Allocator is responsible for keeping a list of available free blocks. It decides which of the free block is to be assigned next according the nature of data, as discussed in detail in section 3.3. Victim block collector selects the victim block by considering the utilization; migrations cost and erase counts of candidate blocks. Finally, Cleaner reclaims invalid pages of victim block to generate new free space, as defined in section 3.4.

The reclamation takes place either in the background when CPU is idle or on-demand when the amount of free space drops below the predetermined threshold. However, the prediction of the I/O workload such as the number of I/O request arrivals during the next garbage collection execution can control the number of victim blocks to be erased according to the estimated I/O workload, as in [8, 12].

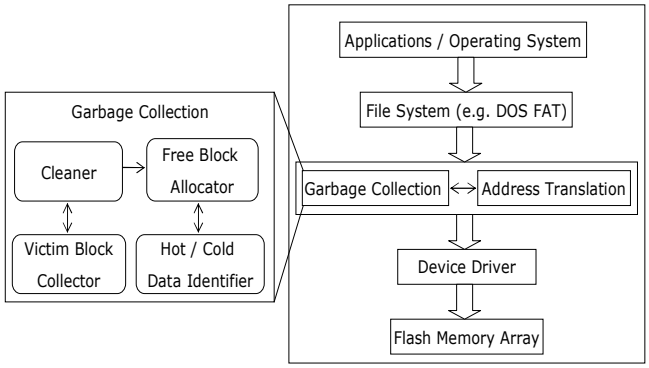

Figure 1. Proposed system architecture.

3.2. Flash Meta Data Structure

The log based schemes maintain the lists of data blocks, log blocks, and free blocks in map blocks, as in [9, 11]. The data blocks hold ordinary data and the log blocks are used to store update writes to data blocks. In this paper, we additionally propose the dirty blocks list that is used to save the information related to obsolete blocks, and we consider blocks by nature of data stored in blocks, either hot or cold. We identify nature of data by hot/cold data identifier. The hotness denotes how often the block data is modified. Hence, hot blocks are frequently updating blocks, and cold blocks are infrequently updating and read only blocks. Data blocks are logically divided in hot data and cold data blocks. Initially each data block defines as the hot block. When a hot data block becomes read only or updates infrequently, its nature changes to cold. Similarly log blocks are divided in hot log and cold log blocks. We assign log blocks by separate log block scheme [14], where hot and cold both types of data blocks have separate limited number of log blocks. To overcome the scanning of whole flash array on every time of booting, we are using the limited number of map blocks to store the meta-information.

3.3. Block Allocation Policy

The hot and cold data can be separately stored by carefully selecting the next fresh block to be used from free blocks pool. In this paper, block allocation is applied by the free block allocator based on degree of hotness of data. In case of cold, data and log block, new block with high erase count is allocated. This is because cold data is either read only data or updates infrequently. That's why the new block experiences less invalidation. Correspondingly, In case of hot, data and log block, a block with low erase count is selected to increase its usage. The new block can be expected to reach its next cleaning cycle earlier because of its frequently updating nature of data. This phenomenon keeps balanced the erase cycles on all blocks.

3.4. Garbage Collection Policy

The garbage collection process is consisted of the rewrite and the erase operations. There are usually three stages involved. System first selects victim block and identifies valid data that are not obsolete in the victim block. System copies the valid data from victim block and rewrites the data to the new physical location of memory, called data migration. Finally, the victim block is erased and available for new data. Cleaning efficiency depends mainly on the rewriting phase, where data migration cost highly impacts on total cleaning cost. Erasing phase consists only of a hardware operation, erase, which incurs a fixed cost. Therefore, for efficient memory utilization and reduced cleaning cost, the effective victim block selection is highly important.

In this paper, the first priority for erasure is for already obsolete blocks available in dirty blocks pool. We achieve reduced erase operations as well as efficient wear-leveling by not erasing blocks immediately after they become obsolete. Blocks are collected in dirty blocks pool and when system triggers the cleaner for free space, block with least ECN is selected for erasure, then provides to free

blocks pool. In other case, if dirty blocks pool is empty, the Victim Block Collector applies victim selection. We prefix two main goals for our victim block selection policy.

Goal 1: select blocks with optimal memory bandwidth utilization.

System selects the blocks where margin of used pages is more than free pages from the total number of used blocks. We aim not to waste the expensive free space. Therefore, we select the blocks with optimum written pages.

Goal 2: select blocks with least migration cost.

System extracts the blocks those have minimum number of valid pages with respect to total number of used pages to save the cost of copying valid data from victim block to new memory location.

We substitute above two goals with equation 1, where the "UtBlock" represents the utilization of block that is total number of written pages, and "MC" shows the migration cost of number of valid pages in candidate block "n". We set priority to the usage of block on migration cost by $0 \leq \beta < \alpha$ constant determinants.

$$Victim\ Score(n) = \alpha \times UtBlock(n) - \beta \times MC(n) \quad (1)$$

In this paper, garbage collection activates in two directions. First, when assigning the new log block crosses the predefined limit, system triggers the garbage collection on log blocks. System selects the victim log block which has highest victim score by equation 1, and then it reclaims by Split operation [14]. Split operation copies up-to-date pages from victim log block to newly allocated log block. After copying all the valid pages, the former log block is returned to the dirty blocks pool for erasure, and further updates to data blocks forward to new log block.

Second, when system crosses the threshold *N* of free space, it triggers the garbage collection on data blocks for system reliability. The threshold value *N* is not an independent variable but is fixed as the maximum allowable utilization of the media. Equation 1 applies on both natures hot and cold of data blocks and the block with maximum victim score is selected as victim data block, and then it reclaims by merge operation. If equation 1 achieves more than one candidate blocks with same victim score value than block with least ECN is selected for erasure.

We explain our garbage collection policy on data blocks by an example, as shown in Figure 2. Assume there are total eight data blocks in use, from Physical Block Number (PBN) 1 to PBN8, and every block is composed of four pages. Victim score is calculated by equation 1 with α=1 and β=0.5, as shown before every block. Therefore, the PBN7 is selected as victim block with highest victim score value by fulfilling both predefined goals as there is no free page that could be waste uselessly because of erasure operation and it has less number of valid pages than the obsolete pages that saves the migration cost.

Compare to previously proposed block recycling greedy policy [16], our victim block selection policy sets the priority on bandwidth usage of candidate blocks than data migration cost. Therefore, if we select block by greedy policy than the possibility for selection of PBN1 and PBN2 is likely higher than PBN7 because greedy policy selects the victim block in first-in-first-out order with least number of data migration regardless of how many free pages can be reclaimed. Therefore, our victim block selection policy is fully associative with memory space utilization and migration cost saving.

After selection of victim block, the merge operation is applied, where the valid data from victim data block PBN7 and its corresponding log blocks PBN9 and PBN10 copies to new allocated data block PBN11, as shown in Figure 2. New block is assigned by the free block allocator regarding victim data block nature for efficient data redistribution. After merge operation, victim block is erased and provided to free blocks pool.

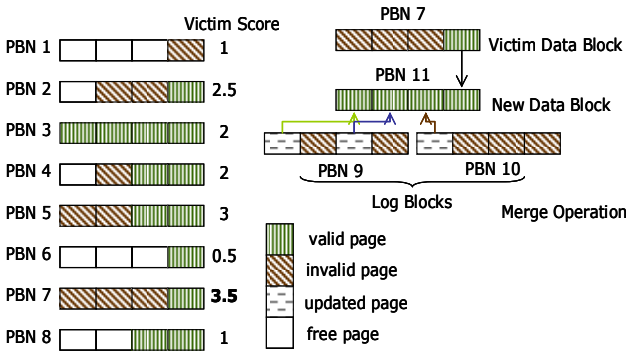

Figure 2. Garbage collection on data blocks.

We achieve efficient utilization of storage media by selecting the optimal used block for erasure with reduced data migration. Our proposed cleaning scheme efficiently derives the approach to maintain the criteria for victim block selection and data redistribution.

3.5. Hybrid Wear-Leveling Policy

A good wear-leveling policy evenly distributes the erase cycle counts on all blocks to prolong the life time of flash media. Thus the effectiveness of a wear-leveling policy could be evaluated in terms of the standard variation of the erase counts of all blocks and the earliest appearance time of first worn-out block.

Wear-leveling classifies in two directions, dynamic and static wear-leveling. In dynamic wear-leveling, recycling of blocks only happens to blocks that are free or occupied by hot data. Therefore, cold data is likely to stay untouched for long time regardless of how updates of hot data wear-out other blocks. Static wear-leveling is orthogonal to dynamic wear-leveling.

Its objective is to move any cold data from staying at any block for a long period of time to evenly apply erase count to all blocks. Therefore, it proves that endurance improvement is severely constrained by data nature.

In this paper, we propose hybrid wear-leveling, and aim to evenly wear-down blocks with hot and cold both types of data by following two schemes.

First scheme: we achieve dynamic wear-leveling on every block allocation time as we have already discussed in Section 3.3 that system assigns the high and low ECN blocks to cold and hot blocks, respectively. The derived thought of this approach is that careful free block allocation by considering diverse nature of data certainly evenly-wear the flash media.

Second scheme: we achieve static wear-leveling based on Cease-Swap Policy. The swapping is applied at regular intervals to move cold data from low to high erase count blocks. It prevents high erase count blocks from being overly worn by storing infrequently updating and read only data, and allows low erase count blocks to be used for frequently updating data.

Cease-swap policy: there are irrespective numbers of erase count blocks in hot, cold and free blocks pool. System checks the condition using equation 2, and block with maximum erase count is extracted either from Hot Blocks Pool (HP) or from Free Blocks pool (FP) by using Max_{EC} function. Then condition in equation 3 is examined to determine the need of wear-leveling by given user-configurable threshold TH.

$$Max_{EC}(EC_{HP}, EC_{FP}) \quad (2)$$

$$Max_{EC} - Min_{EC}(EC_{CP}) > TH \quad (3)$$

If the difference between a maximum erase count block from hot or free blocks pool and a minimum erase count block from cold blocks pool (CP) is more than given threshold TH than the following procedure is performed.

Step 1: valid data from $Min_{EC}(EC_{CP})$ and from corresponding log blocks is merged to new allocated free block with high erase count.

Step 2: erase block $Min_{EC}(EC_{CP})$.

Step 3: make available recently erased block to free blocks pool.

Whenever the condition in equation 3 becomes true, it is assumed that on the one side block $Min_{EC}(EC_{CP})$ have not been erased for a long period of time because of storing cold data, and on the other side block Max_{EC} have been erased plenty of time because it frequently stores hot data. Therefore, the cold data is merged on high erase count block available in free blocks pool. After erasure, the block from cold data list moves to free blocks pool to start being worn.

Traditional Dirty-Swap operation takes at least six steps to move cold and hot data in high and low erase count blocks respectively, as presented in dual-pool scheme [2]. This approach takes two expansive blocks erase, two blocks copy, and extensive meta-information change to commit one swap operation. Also many other previous schemes like [16] consume large size of main memory for data buffering while swapping. However, our proposed Cease-Swap Policy takes one block erase, one block copy, comparatively small changes in meta-data, and no space required in main memory for data buffering while swapping.

Another reason to avoid the traditional movement of data is that as the hot blocks become dirty frequently and next time block allocation selects the low erase count block from free blocks pool automatically. So there is no need to move hot data in low erase count block during swap. Thus, the high erase count block from hot blocks pool systematically saves in free blocks pool after erasure, and is assigned to cold data on next time block allocation.

The proposed hybrid wear-leveling policy efficiently evenly wear-down, hot and cold, both natures of blocks by applying separate associative methods. Our proposed scheme is consistent with changing access patterns of data, and it saves expansive erasure and data migration cost on every time data movement.

4. Performance Evaluation

In this section, first, we establish the simulation environment, and then experimental results are presented and discussed.

4.1. Simulation Environment

To evaluate the performance characteristics of our proposed garbage collection and wear-leveling scheme, AMI, and previous schemes as greedy policy [16] and dual-pool algorithm [2], we developed a simulator and performed trace-driven simulations. We have built a simulator with 1 Gigabyte 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 unit. Every page size is 512 bytes. We use 15μs for a page read, 200μs for a page write, and 2ms for a block erase from [15] product.

We use five data traces in this experiment, see Table 1. These traces have been obtained from the author of [11]. Traces A, B and C are generated from digital cameras and thus contain both small random inputs and frequent large sequential inputs. Traces D and E contains many random inputs and infrequent large sequential inputs. As the flash memory is being used as the storage media for more general computer systems including laptop computers [10], we believe that these traces are complex enough to show the characteristics of our proposed scheme.

Table 1. Traces used for simulation.

Trace	Workload Description	Number of Inputs
A	Digital camera (A company)	4,618
B	Digital camera (B company)	5,111
C	Digital camera (C company)	69,576
D	Linux O/S	18,900
E	Symbian O/S	4,049

For each given trace, simulator counts the numbers of reads, writes and erases operations, and calculates the number of consumed blocks and size of device utilization, and also determines degree of worn by total number of allowed erase cycles per physical block. To prove the enhancement of our idea for large size of systems, we execute every trace file for large number of times. Finally, we provide the results of every execution till 10,000 times.

We have discussed in section 3.4 that garbage collection triggers on data blocks when the number of free blocks crosses a threshold N. For our experiment, we completely fill flash memory for effective media utilization. Thus, we always keep only one free block for reliable cleaning process. In order to achieve wear-leveling, the user-configurable parameter TH is discussed in section 3.5. For our simulator, we consider TH as an average erase count value from hot data blocks pool for both AMI and dual-pool schemes.

4.2. Simulation Results

This section describes the experimental results. Here, we analysis and compare our scheme AMI with the previous schemes [2, 16]. To have fair evaluation, all three schemes are simulated in same environment. Our results are presented from Figures 3 to 7, where X-axis denotes the traces symbols, as described in Table 1, and Y-axis denotes the number of blocks occupied, flash space consumption, number of erase and average erase operations preformed due to cleaning policies, and overhead ratio due to wear-leveling schemes, respectively.

Figure 3 presents the results of number of used blocks for all five given traces, where, every trace file executed 10 times. Simulation performed when flash memory was completely free. The repetition of traces results that greedy policy returns in large number of memory blocks consumption than our proposed scheme, AMI. We believe that space consumption is highly depends on storage management policy. The major impact of high performance of our proposed scheme relates to the use of limited number of log blocks. Therefore, greedy policy does not use log blocks and applies sequential distribution of data.

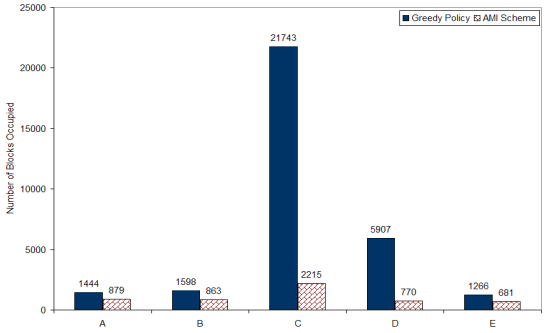

Figure 3. Number of blocks occupied when traces execute 10 times.

Figure 4 shows the results of flash media consumption in Mega-Bytes (MBs), while executed all five traces for 100 times. Our proposed scheme AMI outperforms greedy policy for efficiently utilization of media bandwidth because of our effective data management technique. We specially consider data by its nature, and allocate memory by intellectual understanding of diverse data access patterns. Hence, greedy policy does not consider the write access patterns, and every time the repetition of traces and cleaning process, mix-up the hot and cold data. That result in high erase operations and more consumption of media bandwidth.

Figure 5 shows the results of number of erase operations performed in the unit of thousand, when all five traces executed for 5000 times. In our experiment, no erase operation performed for our scheme AMI till the execution of traces for 1000 times, where only trace-A starts to experience small number of erase operations. But, by greedy policy, memory started to worn with the execution of the traces for 25 times for trace-C.

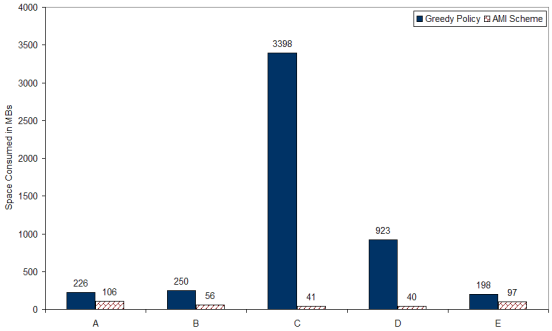

Figure 4. Media consumption in MBs when traces execute 100 times.

Results in Figure 5 clearly show that very low number of erase operations attempted by our scheme AMI compare to greedy policy. Even though, trace-A and trace-E have smaller number of inputs compare to trace-C but they experience more erasures than other traces by our proposed scheme. This experiment shows that trace-A and trace-E contains hot data, and trace-C carries more read only or cold data, and trace-B and trace-D having cold or semi hot data.

For greedy policy, the reason of large number of erases is that after a number of cleaning operations, cold data becomes mixed with non-cold or hot data, under high localities of access. After that time, cold data moves around uselessly together with hot data. For this reason, the utilization of cleaned blocks remains stable at a high value and the amount of free space collected becomes small and cleaning cost increases.

In Figure 5, we can see that by greedy policy trace-C experiences more erase operations than other traces because it is having more write inputs and more cold data. On other side, by our proposed scheme AMI, for same trace-C, there is no erasure at all. The major reason of high performance of our proposed scheme is found as the usage of separate data and log blocks for both natures of hot and cold data. The separate space allocation for both types of data stops mix-up the hot and cold data and results in effective utilization of the media and minimize the erasure operations. We found very surprising and encouraging results, where trace-C not experiences any erase operation even by execution of traces for 10,000 times.

Figure 6 shows the results of number of average erase cycles distributed on every block of media, when the traces files executed 10,000 times. Results clearly show that greedy policy highly suffers its performance compare to our scheme AMI, because of its sequential uniform distribution of data and its first-in-first-out block cleaning strategy. In our experiment, greedy policy experienced 95% more erase cycles than our proposed scheme AMI.

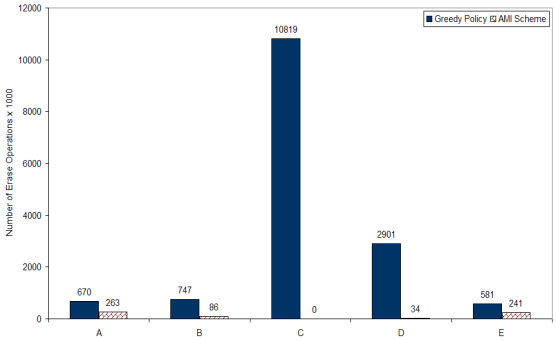

Figure 5. Number of erase operations when traces execute 5000 times.

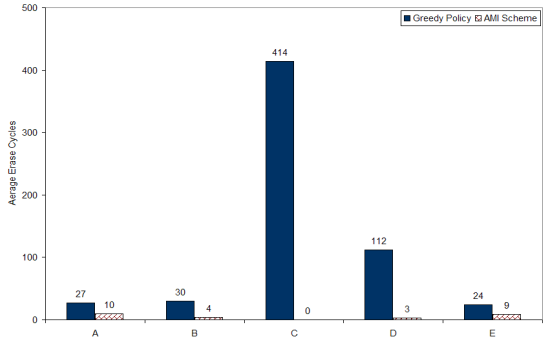

Figure 6. Number of average erase cycles when traces execute 10000 times.

Figure 7 shows the overhead ratio in the unit of milliseconds (ms) appeared by AMI scheme and dual-pool scheme, when the traces files executed 10,000 times. The overhead ratio stands for the ratio of amount of traffic system experiences due to cleaning process as extra writes and erases performed for applying wear-leveling policy. We calculate cleaning cost by using equation 4.

$$Cleaning\ Cost(vb) = Num_{vb}\,(ErC(vb) + MC_{vp}\,(vb)) \qquad (4)$$

The cleaning cost is combination of fixed erasure cost "ErC" and migration cost "MC" of number of valid pages "vp" from number of victim block "Num_{vb}" to new allocated free block. Results in Figure 7 clearly show the effectiveness of both schemes. We observed that dual-pool scheme also performs well and media becomes evenly worn like our proposed scheme AMI, but dual-pool algorithm suffers by the acquired cleaning cost and extra computation. We believe that the wear-leveling scheme performance highly depends on data management policy. However, there is no exact definition given for storage management in dual-pool algorithm; thus, we simulated both wear-leveling schemes on same log based data management platform of our proposed scheme AMI.

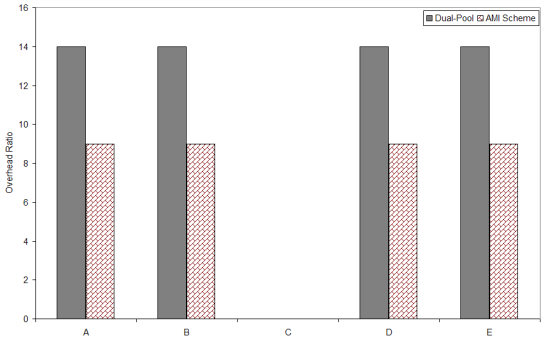

Figure 7. Overhead ratio (ms) when traces execute 10000 times.

In our experiment, dual-pool algorithm experienced 36% more cleaning cost than our proposed hybrid wear-leveling policy. Therefore, our hybrid wear-leveling approach has evenly wear-down all blocks with enhanced system performance. Results show that applying an intellectual and effective endurance management policy associative with diverse nature of data reduces the reasonable cleaning overhead and increases the device life time.

Our proposed scheme AMI outperforms greedy policy and dual-pool algorithm in all cases. However, greedy policy have its own phenomena regarding cleaning and it could give better results while working together with efficient data management policy.

AMI is proving as an efficient technique of garbage collection and wear-leveling for flash devices by giving very encouraging results related to erasure attempted even after a large number of times traces

execution. Results show that our proposed scheme AMI performs outstanding to handle changing data access patterns with time, and highly improves the overall system performance, and prolongs system life time. This is the core achievement of our proposed research.

5. Conclusions

We have presented the data organization and endurance management techniques, to improve overall system performance, called Allocation of Memory Intellectually (AMI). With proposed new system architecture, scheme classifies data according to their write access frequencies, in hot and cold data nature, and improves the space utilization by allocating separate limited number of log blocks to both types of data blocks. Scheme enhances the system life time by managing the blocks according to their degree of worn.

Our proposed cleaning scheme achieves a block to erase with optimal number of space utilization and minimum overhead of data migration. We proposed a hybrid wear-leveling mechanism that is combination of both types of wear-leveling approaches, dynamic and static. In examining the degree of wear-leveling and exploring the effect of flash memory utilization, the proposed method is amazingly performed well.

Performance is evaluated by trace-driven simulations to explore in detail the impact of data access patterns, consumption of device, victim block selection, data redistribution and ratio of erase operations performed. We found very surprising and encouraging results concerned endurance management and significantly very low erasure attempts with amazingly evenly wear-down the flash space. Flash memory life time is thus extended. We improved the overall endurance of flash memory with enhanced system performance.

Acknowledgements

We wish to thank Mr. Muneer Ali Shah Rizvi, professor and dean, Greenwich University, Karachi Pakistan, and Mr. Syed Jamal Hussain, assistant professor, University of Karachi, Pakistan, 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] Ban A., "Wear Leveling pf Static Areas in Flash Memory," *US Patent 6,732,221*, 2004.

[2] Chang P., "On Efficient Wear Leveling for Large Scale Flash Memory Storage Systems," *in Proceedings of the ACM Symposium on Applied Computing*, pp. 1126-1130, 2007.

[3] Chang P. and Kuo W., "Efficient Management for Large Scale Flash Memory Storage Systems with Resource Conservation," *Computer Journal of ACM Transactions on Storage*, vol. 1, no. 4, pp. 381-418, 2005.

[4] Chang H., Hsieh W., and Kuo W., "Endurance Enhancement of Flash-Memory Storage Systems: an Efficient Static Wear Leveling Design," *in Proceedings of the Annual ACM IEEE Design Automation Conference*, pp. 212-217, 2007.

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

[6] Chung S., Park J., Park S., Lee H., Lee W., and Song J., "A Survey of Flash Translation Layer," *Computer Journal Of Systems Architecture Embedded Systems Design*, vol. 55, no. 5, pp. 332-343, 2009.

[7] Chung S. and Park S., "STAFF: A Flash Driver Algorithm Minimizing Block Erasures," *Computer Journal of Systems Architecture*, vol. 53, no. 12, pp. 889-901, 2007.

[8] Han Z., Ryu Y., Chung S., Lee M., and Hong, S., "An Intelligent Garbage Collection Algorithm for Flash Memory Storages," *in Proceedings of International Conference on Computational Science and its Applications*, US, pp. 1019-1027, 2006.

[9] Kwon J. and Chung S., "An Efficient and Advanced Space-management Technique for Flash Memory using Reallocation Blocks," *Computer Journal of IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 298-303, 2008.

[10] Lawton G., "Improved Flash Memory Grows in Popularity," *Computer Journal of IEEE*, vol. 39, no. 1, pp. 159-163, 2006.

[11] Lee W., Park J., Chung S., Lee H., Park S., and SongJ, "A Log Buffer Based Flash Translation Layer Using Fully Associative Sector Translation," *Computer Journal of ACM Transaction on Embedded Computing System*, vol. 6, no. 3, pp. 178-181, 2007.

[12] Longzhe H., Yeonseung R., and Keunsoo Y., "CATA: A Garbage Collection Scheme for Flash Memory File Systems," *in Proceedings of Lecture Notes in Computer Science*, pp. 103-112, 2006.

[13] Rosenblum M. and Ousterhout K., "The Design and Implementation of a Log-Structured File System," *Computer Journal of ACM*

Transactions on Computer Systems, vol. 10, no. 1, pp. 26-52, 1992.

[14] 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*, pp. 125-129, 2005.

[15] Samsung Electronics, "NAND Flash Memory," *KBE00S003M-D411 Data Book*, 2009.

[16] Wu M. and Zwaenepoel W., "eNVy: A Non-Volatile, Main Memory Storage System," *in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 86-97, 1994.

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. Her current research interests include flash memory storages, database systems, and information systems.

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.