A Group-Based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems*

Dawoon Jung, Yoon-Hee Chae, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee
Computer Science Division
Korea Advanced Institute of Science and Technology (KAIST)
335 Gwahangno, Yuseong-gu, Daejeon 305-701, South Korea
{dwjung, yhchae, heesn}@calab.kaist.ac.kr {jinsoo, joon}@cs.kaist.ac.kr

ABSTRACT

Although NAND flash memory has become one of the most popular storage media for portable devices, it has a serious problem with respect to lifetime. Each block of NAND flash memory has a limited number of program/erase cycles, usually 10,000–100,000, and data in a block become unreliable after the limit. For this reason, distributing erase operations evenly across the whole flash memory media is an important concern in designing flash memory storage systems.

In this paper, we propose a memory-efficient group-based wear-leveling algorithm. Our group-based algorithm achieves a small memory footprint by grouping several logically sequential blocks and managing only the summary information for each group. We also propose an effective group summary structure and a method to reduce unnecessary wear-leveling operations in order to enhance the wear-leveling performance. The evaluation results show that our group-based algorithm consumes only 8.75% of memory space compared to the previous scheme that manages per-block information, while showing roughly the same wear-leveling performance.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*Garbage collection*; B.3.2 [**Memory Structure**]: Design Styles—*Mass storage*

General Terms

Design, Algorithms, Performance

Keywords

Flash Memory, Storage Systems, Wear Leveling, Embedded System

*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-C1090-0603-0020).

CASES'07, September 30–October 3, 2007, Salzburg, Austria.

1. INTRODUCTION

NAND flash memory has become one of the most popular storage media for portable devices due to its non-volatility, solid-state reliability, small and lightweight package, and low-power consumption. Many manufacturers are currently deploying flash memory cards and MP3 players with more than several gigabytes of flash storage in the market.

Unlike standard block devices, flash memory media need to be erased before subsequent write operations. An erase operation is performed in a unit of a *block*, while read/write accesses are handled in a unit of a *page*. Since a block is composed of 32–128 pages, flash memory results in additional read and write accesses to keep live data in a block when the block is erased. In addition, erase operation takes much longer time than read or write operation. For this reason, flash-based storage systems generally adopt a thin software layer, called FTL (Flash Translation Layer) [3, 6, 5, 8], to provide the standard block device interface, hiding the presence of erase operation.

Unfortunately, each block in flash memory has a limited number of erase/write cycles and data in a block become unreliable if the block reaches the limit. The current limit for SLC (Single-Level Cell) NAND flash memory is around 100,000 erase/write cycles. The lifetime of flash memory storage can be prolonged if each block in flash memory is updated uniformly. In real systems, however, there is locality in storage access patterns, revealing frequently updated *hot* data and rarely updated *cold* data. Uneven distribution of erase cycles caused by hot data shortens the lifetime of flash-based storage system. To make matters worse, the new generation of NAND flash memory architecture, MLC (Multi-Level Cell) NAND, has higher density, but the endurance cycle is degraded by an order of magnitude compared to the SLC NAND architecture.

Many researchers have proposed various wear-leveling algorithms to address this problem [2, 4, 6, 7, 9, 10, 11]. The goal of the wear-leveling algorithm is to expand the lifetime of flash memory by distributing erase operations evenly across the whole flash memory media. One of the most common approaches is to maintain *cleaning index* for each block and use this information to move hot data into less worn blocks (*young blocks*) or cold data into more worn blocks (*old blocks*). The cleaning index is usually calculated based on such factors as erase cycle, age, and utilization.

Although this approach is simple and effective, it requires memory space proportional to the number of blocks to maintain per-block information. As the flash memory technology advances, the amount of required memory can be significant

due to the increased flash memory capacity. Especially, a small memory footprint is considered an essential requirement for small flash controllers in order to lower the unit cost. K-Leveling [10] lessens this problem by keeping only the difference (*level*) from the least worn block instead of maintaining the full erase cycle for each block. However, K-Leveling still utilizes per-block information.

This paper proposes a memory-efficient group-based wear-leveling algorithm for large-capacity flash memory storage systems. To achieve low memory consumption, our algorithm groups logically adjacent blocks into a single group and only the summary information for each group is maintained. We also suggest an efficient representation for the summary information. Our simulation results show that the proposed algorithm consumes only 8.75% of memory space compared to K-Leveling, while showing roughly the same wear-leveling performance.

2. GROUP-BASED WEAR LEVELING

2.1 Basic Hot-Cold Swapping Algorithm

Block-mapped FTLs [1, 8] are widely used for large-capacity flash storage because they require relatively smaller memory size than page-mapped FTLs. In a block-mapped FTL, LBAs (Logical Block Addresses) are translated into physical addresses in flash memory. Each LBA is divided into a logical block number and an offset. The logical block number is mapped onto a physical flash block (data block) with the assistance of address translation table, and the offset is used to find the page inside the block which contains the designated data. In order to update data, an update block (sometimes called *replacement block* or *log block*) is assigned to the corresponding data block, and incoming data are written in the update block. When there are not enough free update blocks to allocate, a merge operation is triggered to reclaim free blocks by merging a data block and the associated update blocks.

Our group-based wear-leveling algorithm basically relies on *hot-cold swapping* [4, 9, 10] in block-mapped FTLs. The hot-cold swapping algorithm tries to balance erase cycles by periodically swapping hot data in old blocks with cold data in young blocks. We define data blocks with relatively low erase cycles as *young blocks*, while data blocks with relatively high erase cycles as *old blocks*. Since the data in a data block imply that they are not updated for a long time, we regard them as *cold data*. On the other hand, update blocks are likely to be re-written and erased soon due to temporal and spatial locality in storage access patterns. Therefore, the data in update blocks are classified into *hot data*.

In the proposed scheme, the swapping condition is examined whenever a new update block is allocated to a data block. As we described above, an update block will be erased soon and the erase cycle of the block will grow faster than blocks containing cold data. Before an update block is allocated, the erase cycle of the update block is compared with that of the youngest block. Swapping is performed when the difference between the two erase cycles is greater than a certain threshold, *TH*. Rather than explicitly swapping hot data in an old block with cold data in a young block, the algorithm only moves cold data into an old block, thus reducing the number of flash memory operations caused by swapping. Note that the proposed scheme can be also used with other block-mapped FTLs.

2.2 Group-Based Algorithm

The main purpose of the group-based algorithm is to reduce memory requirement for storing wear information. The basic idea is to organize several blocks into a group and to keep only the summary information for each group in memory, instead of maintaining wear information for every block.

In the proposed algorithm, a group consists of a fixed number of adjacent logical blocks, and a group summary represents its overall wear status. The group summary information is used to detect uneven wearing and to find victim groups for wear leveling. Suppose that a group summary shows that blocks in the particular group are much younger than other blocks. Then, the group is selected as a victim group for wear leveling, and a block in the group is swapped with the other old block.

Since the proposed algorithm relies on the group summary information, the effectiveness of wear leveling depends on the accuracy of the summary information. This is because, unlike other wear-leveling schemes that utilize per-block information, the group summary information only denotes the overall wear status of blocks that belong to the group. In order to minimize negative impact on wear leveling, the summary should be designed to represent the wear status of a group as closely as possible.

One of the simplest approaches is to use the average erase cycle of a group as the group summary. This is straightforward and easy to implement; the average can be recalculated whenever a block is erased. For example, if one of the erase cycle is updated from ec to $ec\prime$, the average, avg, would be $(avg \times N + ec\prime - ec)/N$, where N is the number of blocks in a group. However, this simple averaging scheme is not accurate enough to express the wear status of a group. To illustrate, suppose that several blocks in a group are swapped with old blocks. In this case, the average erase cycle of the group gets much higher than young blocks in the group, and those young blocks may not be eligible for swapping any more due to the increased average erase cycle.

In order to enhance the accuracy of the summary information, we maintain two average values, namely a *total average* (AVG_T) and a *partial average* (AVG_P), for each group. The total average denotes the average erase cycle for all the blocks in a group, while the partial average refers to the average erase cycle for only those blocks that are not swapped yet. Initially, AVG_T and AVG_P have the same value. When a young block is swapped with an old update block, AVG_T and AVG_P are recalculated as follows:

$$AVG_T = AVG_T + (EC_{old} - EC_{young})/N \quad (1)$$

$$AVG_P = ((AVG_P \times n) - EC_{young})/(n-1) \quad (2)$$

EC_{young} and EC_{old} represent the erase cycle of the young block and that of the old update block, respectively. N is the number of blocks in a group and n denotes the number of blocks which are not swapped yet. If all the blocks in a group are swapped, AVG_P and n are initialized to AVG_T and 0, respectively. Note that during the calculation of AVG_T and AVG_P, update blocks are not considered.

To check the need for hot-cold swapping, the difference between AVG_P and the erase cycle of a newly allocated update block is compared with respect to TH. If the hot-cold swapping is necessary, the proposed scheme selects the youngest group which has the lowest AVG_P. Since the previously swapped blocks do not contribute to AVG_P, it is

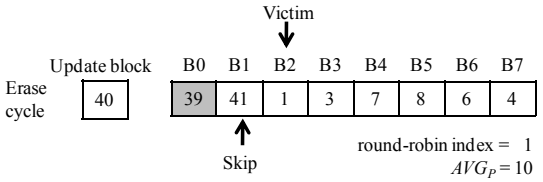


Figure 1: Preventing false swapping

very likely that the youngest group has a block with the lowest erase cycle.

Choosing the actual target block within a group is performed in a round-robin manner. We maintain a *round-robin index* for each group which points to the last selected block in the group. The round-robin index makes it easy to identify whether a block is swapped or not. In addition, the round-robin index can be used for the value of n in Eq.(2).

2.3 Preventing False Swapping

Although the use of AVG_P is effective in selecting the victim group, there may be some unnecessary swapping (we refer it to *false swapping*) since the individual erase cycle for each block is not maintained in memory. Figure 1 illustrates a situation where false swapping can occur. Suppose that $TH = 30$ and the group in figure 1 is selected as a victim group. In this situation, $B1$, which is pointed to by the current round-robin index, becomes the target block for hot-cold swapping. Swapping B1 with a new update block, however, has adverse effect on wear leveling because the erase cycle of B1 is already 41, higher than the erase cycle of the update block.

False swapping not only degrades the effectiveness of wear leveling, but also produces unexpected overhead. Moreover, the chance of false swapping increases as the group size is getting bigger because a small number of blocks with high erase cycles can be easily concealed in a group.

In order to prevent false swapping from happening, we do not solely rely on AVG_P of the group, but makes use of the actual erase cycle for each block. If the difference between the erase cycle of an update block and the actual erase cycle of the target block is less than $(1-\lambda)\times TH$, the target block is skipped and the next block is considered for swapping. λ represents the ratio of the margin in erase cycles with respect to TH. This comparison is repeated until any subsequent block in the victim group meets the condition.

To use the previous scheme, we need a way to obtain the erase cycle of the individual block. We solve this problem by storing the erase cycle of a block in the spare area and by retrieving it during wear leveling. We reserve a three-byte counter in the spare area of the first page in each block. The counter is updated whenever the block is erased. Since the spare area can be written along with the data area using a single write operation, there is little overhead to update the counter. Apparently, reading the spare area accompanies additional overhead. According to our simulation results, this overhead is negligible compared to the cost of false swapping. The use of the counter does not hurt our goal of achieving a small memory footprint as the value of the counter does not stay in memory.

To summarize, our group-based wear-leveling algorithm works as follows.

1. When the FTL allocates a new update block, the youngest group which has the minimum AVG_P is selected as a victim group.
2. In order to examine the wear-leveling condition, the erase cycle of the update block is compared with AVG_P of the youngest group.
3. If the difference is greater than TH, the erase cycle of the block pointed to by the round-robin index is read from the spare area of the block. The erase cycle is then compared with that of the new update block.
4. If the difference is greater than $(1-\lambda)\times TH$, the block is selected as a target block. Otherwise, the block is skipped and the steps 3–4 are repeated.
5. The live data in the target block are copied to the update block, and the address translation table is modified to point to the update block.
6. The young target block is erased and allocated as an update block.
7. Finally, AVG_T, AVG_P, and the round-robin index are recalculated. If the round-robin index reaches the last block in the group, AVG_P and the round-robin index is reset to AVG_T and 0, respectively.

2.4 Determining TH and λ

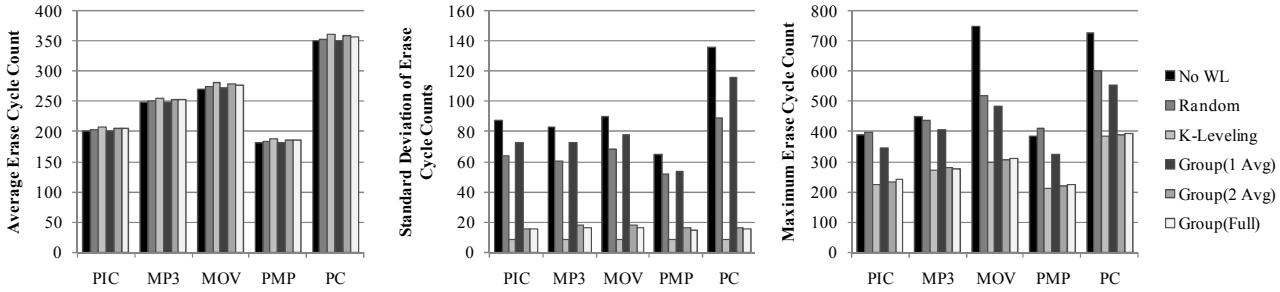
In the group-based algorithm, the threshold value TH controls the degree of wear leveling. This is because the decision whether the hold-cold swapping should be performed or not depends on TH. As TH increases, the algorithm performs less swapping operations and the standard deviation of erase cycles will increase. In contrast, the smaller TH value results in more swapping operations, lowering the standard deviation. Although a smaller TH tends to uniformly distribute erase cycles across the whole flash memory blocks, it is not always a good choice because a too small TH value will cause a lot of swapping operations. In this case, the average erase cycle grows fast and the wear-leveling overhead becomes significant. According to our simulation results, $TH = 30$ shows a balanced performance in terms of the standard deviation, the average erase cycle, and the wear-leveling overhead.

λ is a constant ranging from 0 to 1 and it tunes the degree of false swapping prevention. If λ is 1, the algorithm performs swapping whenever the erase cycle of a block in the victim group is smaller than that of the update block. On the other hand, $\lambda = 0$ means that the erase cycle of a block should be smaller than the update block by at least TH. Hence, the lower the λ value is, the younger block the algorithm tries to find inside a group. Our simulation results show that the values in $0 \leq \lambda \leq 0.2$ perform quite well.

3. EXPERIMENTAL RESULTS

3.1 Methodology

To evaluate the proposed wear-leveling algorithm, we have implemented a trace-driven simulator for large block NAND flash memory. In our evaluation, the log block scheme [8] is used as the underlying block-mapped FTL. For comparison, the random wear-leveling scheme proposed in JFFS2 [11] and K-Leveling are also implemented.



(a) average erase cycle (b) standard deviation (c) maximum erase cycle

Figure 2: The overall results of various wear-leveling schemes

The wear-leveling performance is investigated with the same traces used in [5]. Three traces, PIC, MP3, and MOV, model the storage access patterns of digital cameras, MP3 players, and video players, respectively. The PMP trace models the workload of portable media players (PMPs), where various image files, MP3 files, and movie files are created and deleted. The PC trace is gathered from real user activities on a notebook computer for one week which involve web browsing, word processing, and playing games, mp3 files, and video files. During the simulation, each trace is repeated 50 times to produce a number of erase operations.

The effectiveness of wear leveling is measured using the average and the standard deviation of erase cycles as in the previous work [2, 7]. In addition, the maximum erase cycle among all flash memory blocks is examined, as a block becomes unreliable if it is worn out beyond the limit.

3.2 Overall Performance

Figure 2 compares the overall effectiveness of four wear-leveling schemes: no wear leveling, random scheme, K-Leveling, and our group-based algorithm. In figure 2, these four schemes are labeled as 'No WL', 'Random', 'K-Leveling', and 'Group', respectively. For the group-based algorithm, three different configurations are studied. Two configurations, '1 Avg' and '2 Avg', denote the configuration with a single average and one with two averages (AVG_P and AVG_T), respectively. The 'Full' configuration represents our final group-based wear-leveling algorithm which adds a mechanism to prevent false swapping. In figure 2, λ is set to 0.2 and the group size is fixed as 128.

Figure 2(a) shows that the wear-leveling algorithm produces 1.0–3.5% of additional erase operations. According to the simulation results, K-Leveling induces the largest number of swapping operations, and as a result, it shows the highest average erase cycle. This is because K-Leveling strictly performs swapping operations whenever the difference in erase cycles reaches the threshold, while the others do not.

The standard deviations of erase cycles are illustrated in figure 2(b). K-Leveling and the group-based algorithm represent much less standard deviations than the other schemes. As these wear-leveling algorithms distribute erase operations explicitly by hot-cold swapping, they can achieve good wear leveling. The small difference between K-Leveling and the full version of group-based algorithm comes from the inaccuracy of the group summary information. Even though the erase cycle of a block is smaller than the current update block by more than TH, the presence of the block may be hidden due to the high average erase cycle. For this reason, the number of swapping operations in the group-based algorithm can be decreased, inflating the standard deviation.

We can notice that the group-based algorithm 'Group(1 Avg)' which relies only on the average erase cycle is not effective at all for wear leveling. This is because the average alone cannot properly represent the wear status of a group after several swapping operations. Even if there are young blocks in the group, the average erase cycle of the group cannot satisfy the wear-leveling condition because of a number of old blocks.

The random scheme shows very limited effectiveness. Since the scheme just randomly selects the wear-leveling victim, there can be a lot of unnecessary swapping operations. In addition, the random scheme only triggers wear-leveling operations on one of every 100 garbage collections regardless of the distribution of erase cycles. Therefore, the random scheme does not have much effect on wear leveling.

Finally, figure 2(c) depicts the maximum erase cycle after the simulation. In terms of the maximum erase cycle, the group-based algorithm is almost comparable to K-Leveling. While grouping reduces the number of swapping operations, it is still effective in controlling the erase cycle of the oldest block. The overall overhead of the proposed group-based wear-leveling algorithm is measured to be only 8.3–10.3% of the total garbage collection cost.

Comparing the results of 'Group(2 Avg)' and 'Group(Full)' in figure 2(a) and (b), we can observe that the average and the standard deviation of erase cycles are improved by preventing false swapping. The effect of λ will be analyzed in detail in section 3.4.

3.3 The Effect of Group Size

Figure 3 illustrates the changes in the standard deviation with respect to the group size in the group-based wear-leveling algorithm. As the group size grows, the standard deviation increases since the inaccuracy in the group summary information is getting exaggerated.

There is obvious trade-off between the effectiveness of wear leveling and the size of the summary information that should be maintained in memory. For instance, if we choose a small group size, more memory space is required to store the summary information, while achieving better wear leveling. It is somewhat surprising that the group size of 1024 works quite well, keeping the standard deviation below 20.

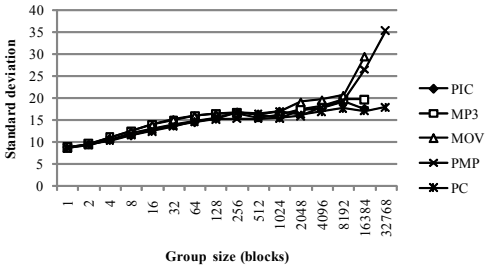


Figure 3: The effect of group size on the standard deviation

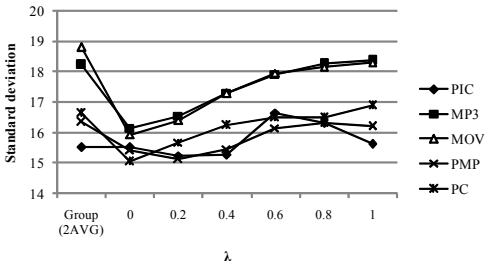


Figure 4: The effect of λ on the standard deviation

3.4 The Effect of False Swapping

Figure 4 exhibits the changes in the standard deviation when we vary λ from 0 to 1. Compared to 'Group (2 Avg)' where false swapping is allowed, we can see that the prevention scheme described in section 2.3 is effective in lowering the standard deviation.

In general, a low λ value tends to decrease the standard deviation, in which case the wear-leveling algorithm aggressively tries to find the block inside a group whose erase cycle differs from the update block by $(1-\lambda)\times TH$. In most cases, the results using $\lambda = 0$ or $\lambda = 0.2$ show the lowest standard deviation.

Note that preventing false swapping has another benefit that reduces the overhead of wear-leveling by eliminating unnecessary flash memory operations. When $\lambda = 0.2$, the wear-leveling cost is cut down by 23.6% on average for the tested workloads. The cost of reading the spare area is responsible for only 0.15% of the total wear-leveling cost since the access time for the spare area is much shorter than other operations. Our simulation results show that 98.9% of swapping operations find proper target blocks in four trials.

3.5 Memory Consumption

Storing erase cycles for all the blocks for wear leveling requires a lot of memory space. At least, we need a 20-bit counter for each block to represent the erase cycle up to 1,000,000. The number of flash memory blocks for 64 GBytes is 512K, and the total memory space required become 1.28 Mbytes. This is simply too big to fit into a small flash controller.

K-Leveling keeps only the difference from the smallest erase cycle [10]. If we assume the threshold K is 30, five bits are required for each block. The resulting size of the wear information is 320 Kbytes ($512K \times 5bits = 320Kbytes$) when the size of flash memory is 64 Gbytes.

The proposed group-based scheme requires two averages for each group. To represent an average ranging from 0 to 1,000,000, we allocate 24 bits for each average (AVG_T and AVG_P). In addition, the round-robin index for each group occupies one byte. As a result, each group requires 7 bytes. If a group consists of 128 blocks, the number of groups is 4,096 for 64 Gbytes flash memory and the total size of the summary information is 28 Kbytes. Compared to K-Leveling, the group-based algorithm requires only 8.75% of memory space. Section 3.3 has shown that the group size of 1,024 also works reasonably well, in which case the required memory is reduced to 3.6 Kbytes.

4. CONCLUSIONS

In this paper, we have proposed a group-based wear-leveling algorithm for large-capacity flash memory storage systems. The algorithm basically relies on hot-cold swapping in block-mapped FTLs. By grouping several logically adjacent blocks into one group and by managing only the summary information for each group, we significantly reduce the memory footprint required for wear leveling. Our evaluation results show that the group-based wear-leveling algorithm provides roughly the same wear-leveling performance, while consuming less than 8.75% of memory space compared to the existing K-Leveling scheme. We plan to investigate other metric for the summary information that is more effective to wear leveling.

5. REFERENCES

[1] A. Ban. Flash file system optimized for page-mode flash technologies. *United States Patent*, (5937425), 1999.

[2] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using data clustering to improve cleaning performance for flash memory. *Software-Practice and Experience*, 29(3):267–290, 1999.

[3] Understanding the flash translation layer (FTL) specification. `http://www.intel.com/design/flcomp/applnots/29781602.pdf`, 1998.

[4] E. Jou and I. James H. Jeppesen. Flash memory wear leveling system providing immediate direct access to microprocessor. *United States Patent*, (5568423), 1996.

[5] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A superblock-based flash translation layer for nand flash memory. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 161–170, 2006.

[6] A. Kawaguchi, S. Nishioka, and H. Motoda. A flash-memory based file system. In *USENIX Winter*, pages 155–164, 1995.

[7] H.-J. Kim and S.-G. Lee. A new flash memory management for flash storage system. In *COMPSAC '99: 23rd International Computer Software and Applications Conference*, 1999.

[8] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, May 2002.

[9] K. M. J. Lofgren, R. D. Norman, G. B. Thelin, and A. Gupta. Wear leveling techniques for flash eeprom systems. *United States Patent*, (6850443), 2005.

[10] S. Park. K-leveling: An efficient wear-leveling scheme for flash memory. In *UKC '05: Proceedings of The 2005 US-Korea Conference on Science, Technology, and Entrepreneurship*, 2005.

[11] D. Woodhouse. JFFS : The journalling flash file system. Ottawa Linux Symposium, 2001.