File System Support for Adjustable Resolution Applications in Sensor Networks*

Vikram Munishwar†, Sameer Tilak‡ and Nael B. Abu-Ghazaleh†

†Department of Computer Science
State University of New York, Binghamton,
Binghamton, NY 13902
{vmunish1,nael}@cs.binghamton.edu
‡SDSC, UC San Diego, MC 0505, 9500 Gilman Drive
La Jolla, CA 92093-0505
sameer@sdsc.edu

Abstract. Flash memory is often the technology of choice for sensor networks because of its cost-effectiveness and attractive energy properties. In storage-constrained sensor network applications, the monitored data is typically stored in multi-resolution fashion. This allows reclamation of some storage space when needed by reducing the quality of stored data by eliminating some of the precision. Existing sensor network file systems are optimized for sequential logging of the data. However, flash memories have a number of unique properties that require careful consideration in file system design. In this paper, we show that in applications where adjustable resolution occurs, sequential logging file-systems result in an inefficient implementation of adjustable resolution. We propose an alternative implementation of the file system where data components are grouped with each other according to resolution. Thus, reducing resolution is implemented by simply erasing the pages with the excess resolution components. We have implemented the proposed scheme on crossbow MICA2 sensor nodes. In addition, using TOSSIM simulations, we show that as compared to the existing approach, the proposed scheme results in significant savings in read and write operations to the flash (thereby in turn saving energy, and reducing wear). Further, we show that wear leveling can be maintained over time by assigning the most significant data to the most frequently used pages.

1 Introduction

Wireless sensor networks (WSNs) are an important emerging technology that can provide sensing at unprecedented resolution. This capability is of importance to a wide range of scientific, military, industrial and civilian applications. However, many battery-operated sensors have constraints such as limited energy, less computational ability, and small storage capacity, and thus protocols must be designed to deal efficiently with these limited resources. In WSNs, available energy is the primary resource that defines the network's useful lifetime. Thus, concern for energy efficient operation permeates all aspects of sensor network design and operation.

There exists a class of sensor networks where the sensed data (or a subset of it) is not relayed in real-time. In such applications, the data must be stored, at least temporarily, within the network, until it is later collected or queried by an observer. Alternatively, the data may cease to be useful and be discarded or compressed to make room for more

* This work is partially supported by NSF grant CNS-0454298 and US Army project W911SR-05-C-0014.

important data. The data may also be used in dynamic queries. Two application classes where the storage management problem typically arises are scientific monitoring applications and augmented reality applications. We briefly overview these applications and reader is encouraged to refer to [16] for further details.

Sensors in an unattended network for a remote scientific monitoring application may collect data and store it for extended periods [9, 12, 18]. Scientists (observers) might periodically visit the network to collect this data. Data collection is not pre-planned: it might be unpredictable and infrequent. However, the data *query* model is simple: data is collected one time. Alternatively, in an augmented reality application, users may dynamically query sensors for the collected data [17]. The queried data can be real-time, recent, or historical data. Therefore, to be able to respond to queries that span temporally long periods, sensors must store data. In these storage-bound networks, the design of the storage system is critical to the overall energy efficiency and performance of the network.

In storage bound sensor networks, it is often the case that the resolution of the collected data will need to be adjusted periodically. For example, in a scenario where a node runs out of storage space, it is desirable to reduce the quality of stored samples to make room for additional data to be collected, rather than discard the new data or overwrite the old data completely due to lack of storage. Similarly, it is possible that data is of interest at the highest resolution only for a period of time; as data ages, low-resolution information may suffice [7]. The data itself may be required at different resolutions in response to different queries. For example, a certain aggregate query may require only the highest resolution component of the data. We call these application *adjustable resolution applications*. Storing data in multi-resolution fashion allows reclamation of some storage space when needed by reducing the quality of stored data by eliminating some of the precision.

For non-volatile storage, flash memory is often the technology of choice for sensor networks and mobile devices because of its cost-effectiveness and attractive energy properties. Section 2 presents some background material related to flash memory organization. A number of research studies have examined various aspects of data storage in sensor networks [11, 15, 7]. Moreover, file-systems for flash memory devices for use in sensor networks have been developed [8, 5], with emphasis on support for sequential data logging (rather than random access). Section 3 overviews these and other related work. Unfortunately, in the case of adjustable resolution applications, the interaction between application and the file-system has received almost no attention. For example, flash memories have properties that require special care from file system designers. Most importantly, flash memory can only be modified (written, or erased) with a page granularity. Thus, if a word needs to be modified in a given page, the whole page has to be flashed (erased) and then rewritten. As a result, random access data modification is a costly operation. In addition, pages can become faulty with usage– a problem called *page wear*. Therefore, it is desired to load balance the number of times each page is written (*page wear leveling*).

Existing sensor network file systems are optimized for sequential logging of the data. We show that in these applications, adjusting resolution under existing file system implementations leads to excessive read and write operations to the flash. These result in high consumption of energy as well as increasing the wear on the flash. Essentially, the data whose resolution is to be adjusted must all be read, then the least significant components are deleted, before the remaining data is stored again. The same argument is true when querying data at different resolutions. In the current implementation, all the data would have to be read and only the relevant resolution data is extracted.

To counter this inefficient operation, we propose an alternative organization where data is organized into pages according to significance. This approach allows reducing resolution by simply erasing the pages with undesired components. Further, querying data by resolution results in only the required data being read. We describe the problem

and present the proposed scheme in Sections 5 and 6 respectively. In Section 7 we present an experimental study characterizing the performance of the proposed scheme. Finally, Section 9 presents some concluding remarks.

2 Background

In this section, we first review characteristics of flash memory devices. We then overview two major limitations of existing flash memories, which have significant impact on the design of flash based file systems. Finally, we overview the organization of the flash and EEPROM device on MICA2 sensor nodes, which were used in our implementation and simulation.

2.1 Flash Memory Devices

Flash memory provides non-volatile storage at a lower cost than traditional storage devices such as magnetic disks. Other advantages of flash memory over magnetic disks include: (i) fast read access time; (ii) low-energy dissipation; (iii) light-weight and small form-factor; (iv) better shock-resistance. Because of these properties, flash memories have become a de-facto storage technology in sensor networks and embedded/mobile devices in general.

A flash memory is similar to Electrically-Erasable Programmable Read-Only Memory (EEPROM) in terms of physical architecture; in fact, flash is essentially a block-writable EEPROM (where conventional EEPROM is byte-writable). Flash memory is broadly categorized into two types: NOR flash memory and NAND flash memory. These names are based on the type of logic gate used in its cell. NOR flash memory is mainly used for storing small data, source code, etc. It has a capability of in-place execution of a code. This is because it behaves like ROM memory, mapped to a certain address [4]. NOR flash is characterized by its faster read time, nearly equal to the read times of DRAM [10]. However, it has a disadvantage of slower write and erase operations. On the other hand, NAND flash memory is mainly used for data storage due to its i) higher density; ii) faster erase and write operations; and iii) longer re-write life expectancy [3]. Since NAND memory is accessed in much similar way as block devices, in-place code execution is not possible. To execute code present on NAND flash, the code must be first taken into memory mapped RAM and executed there. Thus, NAND flash always needs to be associated with Memory Management Unit (MMU) for code execution purpose [4].

Read cycle	80 - 150 ns
Write cycle for one byte	1 - 10 μs
Erase cycle for one block	1 s
Power consumption	30 - 50 mA in an active state
	20 - 100 μA in a standby state

Table 1. Flash Characteristics

2.2 Unique Operational Properties of Flash Memory

In comparison with the traditional memories, there are two different operational properties associated with flash memories. These properties have significant implications on the design of a file system for flash memory. These properties are:

1. Limited number of erase/write operations: Flash memory has finite number of erase/write cycles, from 10,000 to 100,000 cycles. So random writes to flash can cause wearing out of some part of the flash while the other part remaining usable. To reduce such problems, it is desirable to reduce the number of erase/write operations (wear) and distribute them evenly across the pages of the flash, thereby achieving *wear leveling*.
2. Lack of in place data overwriting capability: Flash memory doesn't allow writing data to a page before erasing it. Erase is a costly operation because it needs to be done at block granularity, whereas read and write operations are carried out at page granularity. So in effect, to rewrite a single byte in a flash, we need to read all pages in a block, erase that block, modify the requested page, and rewrite all pages back to the flash. Thus, random updates are costly operations.

2.3 Flash Resources on MICA2 Nodes

In our implementation, we used the Atmel AT45DB041B flash [2] chip present on Berkeley MICA2 motes[1] In this section, we describe some important details of this flash. The total storage capacity of the flash is 512KB. The storage area is divided into sectors, blocks, and pages. Each block contains 8 pages, each of 264 bytes. So the maximum number of pages that can be stored on this flash is 2048. Pages need to be erased before being written. All program operations to the flash take place on page-by-page basis; however, the optional erase operations can be done at the block or page level [2]. This complicates the process of in-place modification (or overwriting) of page directly. Two on-chip buffers, 264 bytes each, are used during read and write operations on flash. Write operations on flash are performed in two phases, in the first phase, data is written to the on-chip buffer until it gets full, and in the second phase, the data in on-chip buffer is subsequently written to the flash memory.

In addition to the flash, 128K Program Memory, 4K SRAM, and 4K EEPROM are also available on MICA2 mote. The EEPROM provides small, additional storage for storing program related data such as data-structures, variables, etc. EEPROM provides a persistent storage with slow access speeds as compared to SRAM. Byte wise reads and writes are possible with EEPROM. With 1 MHz clock, read and write times for one byte are approximately 4 μs and 8 ms, respectively. Though wear leveling is an issue for EEPROM, it is not as severe as flash because it has an endurance of at least 100,000 writes. We use EEPROM primarily for storing our program specific data structures like bitmaps.

3 Related Work

In this section, we first discuss storage related efforts in sensor networks. We then describe flash based file systems.

[1] However, none of the proposed ideas are specific to this flash; they can be generalized to other flash targets directly.

3.1 Storage management in WSN

Recently, storage management in sensor networks has received considerable attention [15, 6, 11]. Tilak et al. [15] studied the problem of using limited persistent storage of a sensor to store sampled data effectively. They proposed a collaborative storage approach that takes advantage of spatial correlation among the data collected by nearby sensors to significantly reduce the size of the data near the data sources. Their approach provides significant savings in storage space and energy. It also performs load balancing of available storage space. Ganesan et al. [7] proposed a similar data management scheme and motivated use of adjustable resolution scenarios. While these works operate at the level of networked storage, this paper focuses on the local storage (storage organization on flash memory available on a given sensor). We believe that in complement to these works, the proposed scheme provides a lower level service, targeting efficient data organization on flash memory for adapting resolution. The aging policy proposed by Ganesan et al. [7] dictates that resolution of data (across the sensor network) is adapted lower as a function of the age of the data. In our approach we apply a similar aging policy but for data stored on local flash.

3.2 Existing file systems

We now describe flash based file systems that exist in resource-rich embedded systems (such as PDAs) as well as sensor networks.

YAFFS and JFFS Existing flash based file systems such as the Journaling Flash File System (JFFS version2) and Yet Another Flash Filing System (YAFFS) are more suitable for resource-rich devices such as PDA. The main reason behind this is their high run-time memory consumption. Therefore, low resources (low computing power, limited memory and storage capability) and different application characteristics demand for design of a new file system for WSNs. We now describe two such efforts in that direction.

Efficient Log-structured Flash file system (ELF) Dai et al. have modified a traditional log structured file system approach to make it adaptable for resource constrained sensor nodes [5]. ELF is an efficient and reliable file system, which also supports general file operations such as create, open, append, delete, truncate, etc. The main idea in ELF is that instead of creating separate log entries for each write-append operation, each log entry is stored in a separate page. This reduces overhead of storing separate log entries even for small write-append operations. In addition they propose using write cache to gather all appends to the same page, to achieve wear leveling. ELF also provides other unique features to the flash file system such as garbage collection, and crash recovery. Like our implementation, ELF makes use of internal EEPROM storage that is available on a sensor node, to store crash recovery data.

Matchbox Matchbox [8] is a simple filing system designed specifically for sensor nodes. It stores meta-data and files as byte streams in memory. First page of meta-data byte stream is called as *first-meta-data-page*. File data is stored in data pages. There is no fixed root page for the file-system, instead, they maintain a version number for pages of type first-meta-data-page, and choose a page with highest version number as a root page. This helps avoid frequent writes to a single page. Another important data structure they use is a bitmap for free pages. It is constructed during the flash traversal at boot time. Search for a free page starts from the last allocated page, which helps achieve basic wear leveling. Due to sequential reads and appends, matchbox doesn't offer an efficient

scheme for handling multi-resolution data. We believe that Matchbox can be extended to incorporate our idea of Resolution-Order Storage organization (ROS) (described in Section 6).

4 Design Goals

In this section, we outline the performance metrics of interest to storage bound applications. The metrics are related to: (1) how efficiently the operations are implemented from an application as well as energy perspective; (2) storage efficiency; and (3) wear and wear leveling. These are described below in more detail.

- **Storage efficiency** It is important to use the available storage such that even if it gets full, the application should have an option of degrading the quality of existing data and allowing storing new data. Again, this should be achieved with very little or no overhead of extra writes on flash.
- **Flash Access efficiency**: The number of pages that need to be read or written for a given operation depends on how the data is organized and indexed. This metric measures the efficiency in implementing the operations required on the flash.
- **Energy efficiency** Since the battery life is the most important factor contributing towards the lifetime of a sensor node, additional costs associated to achieve efficient storage management, for example communication overhead, should be avoided.
- **Page Wear**: Since the pages have a limited expected number of writes before they fail, it is desired to reduce the number of writes generated to the flash in the course of application operation. Typical mean writes to failure rating of current generation flash is 100K erase/write cycles.
- **Page Wear leveling**: In addition to reducing wear, it is also desired to load balance the wear (wear-leveling). Wear leveling results in prolonging the time until flash pages start failing, and provide a more graceful failure behavior for the same number of write operations.

Note that these metrics apply to any storage-bound applications. In our case, we specifically target adjustable resolution applications. In this context, we assume that accessing data with variable resolution and deleting data (adjusting resolution) are common operations. Thus, we focus on measuring the metrics above with respect to the implementation of these operations.

5 Adjustable Storage Problem

In this section, we describe the standard *Sequential Logging* (SL) approach and explain how it performs when storage resolution needs to be adjusted. In the SL approach, data is written to the storage device as the sensor node collects it from its transducer or receives it via the network, as shown in figure 1 (a). When resolution adjustment is needed, some portion of the existing data is compressed by discarding lowest precision coefficients and rewriting the remaining coefficients to the flash. Figure 1 (b) shows the flash organization after resolution adjustment phase.

Major disadvantage of this scheme is that the compression or resolution adjustment phase requires additional cost in terms of extra reads and writes on flash. More specifically, in the first pass, the flash is filled sequentially and data is appended to the end of the flash. When the flash is full, data is compressed by discarding least significant coefficients of each data unit, and rewriting the remaining coefficients to the flash. This makes space available for new data.

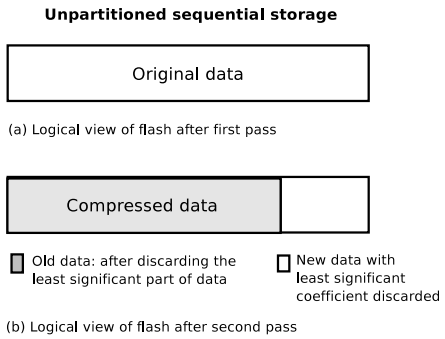


Fig. 1.

6 Proposed Approach

In this section, the proposed Resolution-Order Storage (ROS) organization is described. The discussion is organized into two subsections: the first discusses how the scheme works, while the second discusses wear and wear-leveling issues.

6.1 Resolution-Order Storage (ROS) organization

The key idea in ROS is to store similar precision coefficients together in the same page of the flash. ROS provides an efficient way to retrieve adjustable multi-resolution data, according to the user's requirements. Further, excess resolution may be eliminated easily by erasing the pages with the unneeded coefficients. In this section we discuss the properties of ROS in more detail.

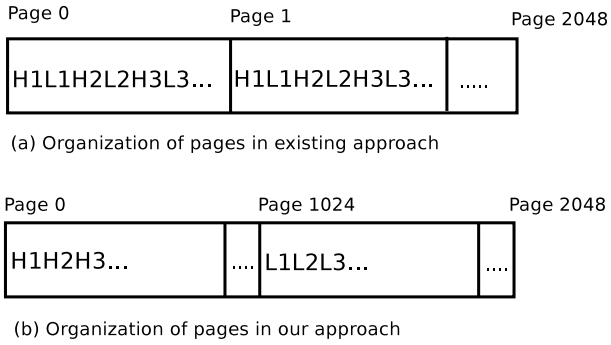


Fig. 2.

Consider an application with two coefficients, High (H) and Low (L). Figure 2 demonstrates ROS comparison between storage organizations of SL and ROS. Figure 2(a) shows how the data is stored sequentially in pages in SL. Figure 2 (b) shows

storage organization of ROS where the first half of the flash stores all higher coefficient (H) components and the second half of the flash stores all lower coefficient (L) components.

We assume that the incoming data is in multi-resolution format; more specifically, it can be partitioned into N coefficients, [C1,C2 . . .Cn], such that the part of data represented by C1 is the highest resolution and the most important coefficient while the part expressed by Cn, represents the lowest resolution and the least important coefficient. The examples of such data include wavelet compressed data, and layered compression data used for multimedia formats [14].

ROS facilitates resolution adjustment: we can just directly overwrite a partition that contains data belonging to least significant coefficient with new data. Thus, resolution adjustment has no cost in terms of read and erase/write operations. The logical storage organization of flash for ROS is shown in Figure 3.

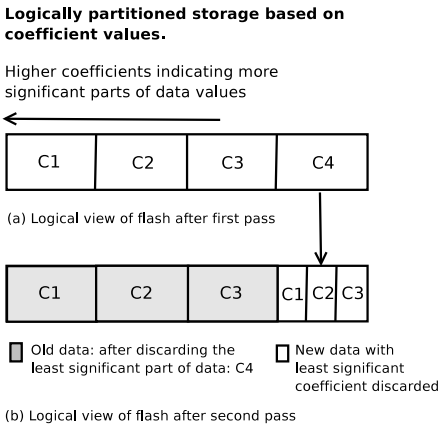


Fig. 3. ROS Organization

6.2 Page Wear leveling

Wear of the flash is a major concern that should be considered while designing storage allocation strategies. A typical flash memory can sustain up to 10,000 to 100,000 erase/write operations per page before the page starts failing [1]. While ROS results in significantly lower average wear than SL (recall, in SL, all pages have to be read, modified and written again), the page use is uneven in a single pass. For example, note that the pages that the initial C_1 coefficients are written to are not erased in the above scenario, while pages where C_4 is initially written to may be erased repeatedly. To avoid early wearing out of individual memory locations, it is important to have a wear leveling mechanism associated with the storage allocation policy for flash.

While wear-leveling within a single scenario is difficult, it is possible to carry out effective wear-leveling across multiple adjustable precision uses of the flash (e.g., every time the base station collects data from the sensor node, data collection starts again with an empty flash). In our implementation, we have used a simple and effective wear leveling policy, which can be applied as pages are allocated. The key idea behind this policy is to track the use count of pages and assign lowest-precision coefficients to the

pages that are written least number of times and vice-versa for the highest coefficients. The intuition is that the maximum number of replacements is done in a part of the flash that was assigned to the lowest-precision coefficients and no replacements are done in the part containing highest-precision coefficient values. As we keep on applying this strategy number of times, the flash tends to get better wear leveling. This intuition is shown to hold experimentally in the evaluation section of this paper.

7 Experimental Evaluation

We simulated the proposed file-system organization on TOSSIM simulator. In addition, a preliminary implementation was carried out and tested on MICA2 motes. The proposed approach, Resolution-Order Storage (ROS), can be applied to storing all types of multi-resolution data, but for this paper, without loss of generality, we are using abstract 32 bit data values, which are adjustable. This data can be degraded in terms of precision by dropping least significant bits. For example, this 32-bit quantity can be a floating point number, represented in IEEE 32-bit floating point format, which allows precision loss by erasing lower bits of mantissa. Since many simple quantities measured by sensor networks, such as temperature and pressure generate floating point numbers, this data type may be considered as a representative of such applications. We varied with different breakdowns of the coefficients. For example, a coefficient distribution of [12, 8, 8, 4] means that the original value was divided into four parts of size 12, 8, 8 and 4 bits respectively.

7.1 Implementation details: Data structures

Our choice of data structures was heavily influenced by the resource constraints in MICA2 motes. By careful selection of data structures, we were able to implement the proposed approach on a MICA2 mote. The major data structures include two bitmaps, one for storing per page coefficient information and the other for storing age information. The width of each entry in bitmap depends on the total number of coefficients to be stored on flash: the higher the number of coefficients, the more the number of bits required to represent a unique coefficient value. For example, for storing 4 coefficients, each bitmap entry will need 2 bits to represent this information.

7.2 On-demand Adjustment Scenario

In the first experiment, we assume the on-demand adjustment scenario. Specifically, data is written until the storage is exhausted; at that time, the current lowest coefficient of all data is removed. The new data can be stored on the reclaimed space until it is exhausted, and the process repeats.

To track energy efficiency, we use the total number of read and write operations performed on the flash as a metric to evaluate both the approaches. The operations are in terms of pages, because the flash can only be operated on in units of pages. Figure 4 shows the number of writes required by both the approaches. The number of writes is significantly lower in ROS than it is in the conventional Sequential Logging (SL). To pick a typical point, for the case of three coefficients, [16, 8, 8], total number of writes performed by SL is 6144 pages, which is almost double the number of writes (3200 pages) performed by ROS.

We now describe about the energy consumption required by using EEPROM for storing bitmaps. Since the sizes of bitmaps grow with the increase in total number of coefficients, it is not possible to store these bitmaps on 4K RAM available on MICA2

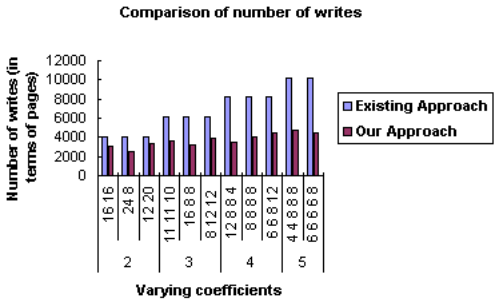


Fig. 4. Write Operations

motes, for an arbitrary number of coefficients. We verified our implementation on MICA2 motes for varying number of coefficients and found that the bitmaps and other variables used in our program can be stored in RAM for only up to 4 coefficients. We addressed this problem by using internal EEPROM of 4K for storing bitmaps when multi-resolution data with more than 4 coefficients needs to be stored. This allows storing bitmaps representing up to 12 coefficient values on EEPROM, leaving RAM to store other program specific variables such as page-buffers, which are used for storing coefficient specific values of incoming data.

Access to the EEPROM takes place at the time of compression, which requires reading coefficient and age bitmaps. This time varies based on the total number of coefficients[2]. We argue that this time should be much less than the time required to read all the data (512KB)[3] and rewriting the compressed data as in SL. The access time can be further reduced by maintaining a separate cache for storing bitmap values for each coefficient in RAM. The filled buffer can be written to the EEPROM at once during the idle time between sampling schedules of sensor(s). In addition, though wear leveling of EEPROM is not a serious problem, it is automatically taken care by the wear leveling scheme in ROS because writes to the EEPROM constitute of updating bitmap entries of corresponding pages, which ultimately takes place in accordance with the writes to the flash.

Table 7.2 shows the total number of reads required by both the approaches. For SL, the number of read operations required increases with the number of coefficients. This behavior occurs in SL because during the compression activity the whole flash is read in order to discard the lowest resolution coefficient of each data value. In contrast, ROS does not require any reads in the resolution adjustment stage.

Finally, we use a metric called *write-cost* to measure the energy efficiency of the implementation. Write-cost is defined as the ratio of total number of bytes read and moved to another place, to the total number of bytes of new data written to the flash [13]. Ideally, the number of bytes written to the flash should be equal to the number of bytes of new data; however, if data has to be compressed and rewritten, then additional operations are required; this metric measures this overhead. Recall that the new approach simply erases the unneeded resolution components and does not require moving data

[2] The time to read coefficient-bitmap, representing 12 distinct coefficient values, and its associated age-bitmap is approximately 8 to 10 ms (with 1MHz clock).

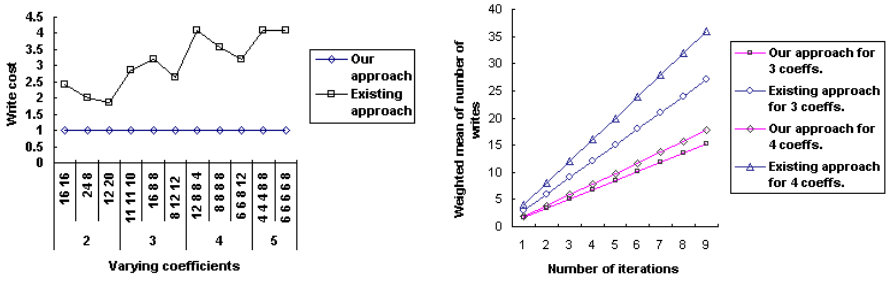
[3] The time required for reading the whole flash of 512K takes approximately 500ms.

Number of coefficients	Coefficient distribution	Seq. Logging	ROS
2	16 16	2048	0
	24 8	2048	0
	12 20	2048	0
3	11 11 10	4096	0
	16 8 8	4096	0
	8 12 12	4096	0
4	12 8 8 4	6144	0
	8 8 8 8	6144	0
	6 6 8 12	6144	0
5	4 4 8 8 8	8192	0
	6 6 6 6 8	8192	0

Table 2. Comparison of Number of Reads

around. Thus, it achieves ideal performance relative to this metric.

$$WriteCost = \frac{|Bytes\ Read + Bytes\ Written|\ of\ old\ data}{|Bytes\ Written|\ of\ new\ data} \quad (1)$$



(a) (b)

Fig. 5. Write-cost and Write distribution study

Figure 5(a) shows that ROS always has a write cost equal to 1.0, as expected. On the other hand, in SL the write cost increases as the number of coefficients increase. This is because, more the number of coefficients, higher the chance of replacing least significant coefficients of old data with new data, and thus higher the overhead for existing approach.

In figure 5(b), we plot values of average wears of all pages in flash, taken after successive iterations of both the approaches, SL and ROS, on flash. Here, one iteration

comprises of filling up the whole flash with data, and keep on compressing it every time the flash is full, to make space available for new data. This process continues until the flash contains only the most significant information. By looking at the graph, it can be easily noticed that the wear increases much faster for SL as compared to ROS.

7.3 Wear leveling

The Figure 6(a) shows that in ROS, the wear leveling of flash improves as the number of iterations increase. During the first iteration, flash memory was statically divided into partitions belonging to different coefficient values. In the successive iterations, highest coefficient value was associated with the pages that were written least number of times and vice versa. To show how the writes are evenly distributed on the flash, we first calculate weighted mean of the number of times a page has been written to the flash. For a given distribution of writes on pages, a1, a2,..., where a1 refers to number of pages written once, a2, number of pages written twice, and so on, the weighted mean W is:

$$W = \frac{\sum_{i=0}^{i=i_{max}} (a_i * i)}{\sum_{i=0}^{i=i_{max}} a_i} \tag{2}$$

Note that the weighted mean indicates the average wear generated to each page in the flash. We then use standard deviation σ in the usual way as:

$$\sigma = \frac{\sqrt{\sum_{i=0}^{i=i_{max}} (a_i * (i - W)^2)}}{\sum_{i=0}^{i=i_{max}} a_i} \tag{3}$$

Wear leveling is estimated using the following metric:

$$WL = \frac{\sigma}{W} \tag{4}$$

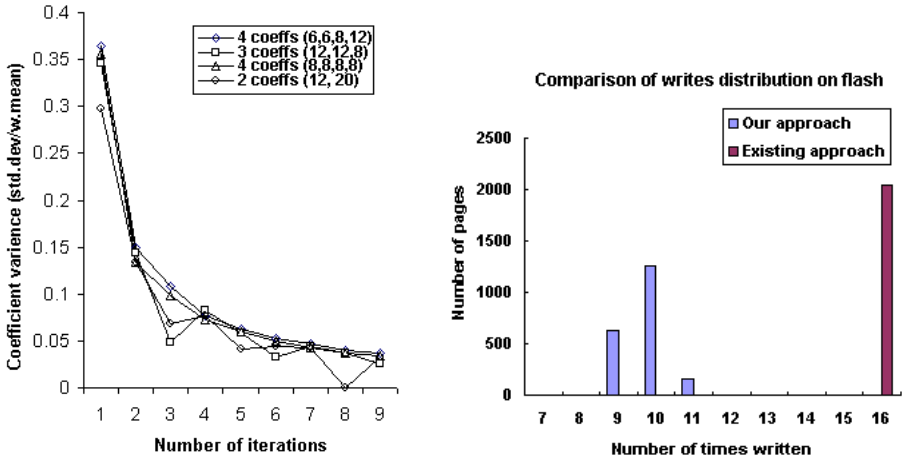
Figure 6(a) shows the WL value after every pass. It can be seen that the WL decreases towards 0 as the number of iterations increases. This shows that the ratio of the standard deviation to the mean is dropping to 0, and there is very little difference in the number of writes to pages within the flash.

Note that for SL, wear leveling is always even, since at the time of compression, the whole flash gets written again. The writes are evenly distributed on the flash at the cost of much higher number of writes.

Figure 6(b) shows a snapshot of write distribution of flash after n iterations, n=4 in the above case. It shows that more than 50% of pages are written 10 times in our approach whereas at the same time, all pages are written 16 times in existing approach.

8 Discussion

In this section we briefly overview the limitations of the proposed approach. The first limitation is that our approach results in higher percentage of reads for random read accesses. This is because, in our approach, each data item is split into multiple components and each of these components are in turn stored on different pages. Therefore, with ROS, reading an entire data item would result in reading N pages, where N is the number of coefficients. In the case of SL, it would simply result in reading of 1 page. However, we argue that random access of data is an uncommon access pattern in an important class of sensor networks applications such as, scientific monitoring applications



(a) (b)

Fig. 6. Wear-leveling study of existing and proposed approaches

(described in Section 1). In these applications, the need for random access of data will not arise since the data needs to be read only once before transferring it to the observer.

The second limitation is an artifact of our implementation. For example, we maintain bitmaps to track coefficient values and age values. The size of these bitmaps is linearly proportional to the number of coefficients. Therefore, as the number of coefficients increases, the size of these bitmaps that need to be maintained in RAM increases proportionately. Therefore ROS has higher run time memory consumption that SL. However, with careful selection and management of data structures, we were able to implement ROS on MICA2 nodes with reasonably small memory footprint (for 4 coefficients, run time memory requirement of ROS was only 3667 bytes).

At present, we assume that the importance of data items is inversely proportional to time, i.e. newer data has higher importance. Therefore, in the proposed scheme, we discard least significant coefficient of old data to create space for new data. In future, we would like to extend our scheme to the cases where the importance of data is application specific. In that case, even the least significant parts of important data values of old data should not be deleted. This can be incorporated in our scheme by extending our page allocation policy in a way that would consider importance of data values. In addition, we will extend the Matchbox file-system to incorporate the proposed approach. We will also consider different data structures along with a wide range of data access patterns and their implications on data organization in flash memory.

9 Concluding Remarks

In this paper, we presented a new storage organization (Resolution-Order Storage) that provides better support for adjustable resolution storage applications for sensor networks. In such applications, the resolution of the stored data may need to be adjusted

(on-demand, or periodically, for example), to make room for new data when the flash space gets exhausted.

Existing file-systems for sensor networks are optimized to sequential logging. Thus, if adjustable resolution is needed, this requires that the whole data be read, the unneeded coefficients removed, then the remaining data re-written to the flash. In the proposed organization, we group similar data coefficients together on the same page. Thus, adjusting resolution can simply be accomplished by deleting the pages that have unneeded coefficients, greatly enhancing the performance relative to the existing organization. Further, the new organization makes accessing data at different resolutions significantly faster (only the required pages are read). This operation is likely to be common in sensor networks with such data.

References

1. Article on flash memory, http://www.bellevuelinux.org/flash_memory.html.
2. Atmel at45db041b flash specifications: http://www.atmel.com/dyn/resources/prod_documents/doc3443.pdf.
3. Characteristics of nand and nor flash: http://www.linuxdevices.com/articles/at4422361427.html.
4. Wikipedia: Article on flash memory, http://en.wikipedia.org/wiki/flash_memory.
5. DAI, H., NEUFELD, M., AND HAN, R. Elf: An efficient log-structured flash file system for micro sensor nodes. In *Proc. ACM SenSys* (2004).
6. GANESAN, D., ESTRIN, D., AND HEIDEMANN, J. Dimensions: Why do we need a new data handling architecture for sensor networks? *ACM Computer Communication Review 33*, 1 (Jan. 2003).
7. GANESAN, D., GREENSTEIN, B., PERELYUBSKIY, D., ESTRIN, D., AND HEIDEMANN, J. An evaluation of multiresolution storage for sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)* (2003).
8. GAY, D. Design of matchbox, the simple filing system for motes, 2003. Version 1.0, August 21, 2003, `http://www.tinyos.net/tinyos-1.x/doc/matchbox-design.pdf`.
9. JUANG, P., OKI, H., WANG, Y., MARTONOSI, M., PEH, L. S., AND RUBENSTEIN, D. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. In *Proceedings of ASPLOS 2002* (2002), ACM Press.
10. KAWAGUCHI, A., NISHIOKA, S., AND MOTODA, H. A flash-memory based file system. In *Proceedings of the 1995 USENIX Annual Technical Conference* (Jan. 1995).
11. RATNASAMY, S., ESTRIN, D., GOVINDAN, R., KARP, B., SHENKER, S., YIN, L., AND YU, F. Data-centric storage in sensornets. In *Proceedings of the First ACM SIGCOMM Workshop on Hot Topics in Networks* (Oct. 2002).
12. A remote ecological micro-sensor network, 2000. (`http://www.botany.hawaii.edu/pods/overview.htm`).
13. ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles* (Feb. 1992).
14. TAUBMAN, AND ZAKHOR. Multi-rate 3-d subband coding of video, 1994.
15. TILAK, S., ABU-GHAZALEH, N., AND HEINZELMAN, W. Collaborative storage management in sensor networks. *International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC) 1*, 1 (2005).
16. TILAK, S., ABU-GHAZALEH, N., AND HEINZELMAN, W. B. Storage management in wireless sensor networks. In *Mobile, Wireless and Sensor Networks* (2005), John Wiley publishers.
17. TILAK, S., MURPHY, A., AND HEINZELMAN, W. Non-uniform information dissemination for sensor networks. In *The 11th IEEE International Conference on Network Protocols (ICNP'03)* (Nov. 2003).
18. VASILESCU, I., KOTAY, K., RUS, D., DUNBABIN, M., AND CORKE, P. Data collection, storage, and retrieval with an underwater sensor network. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems, pp. 154–165, ACM Press, New York, NY, USA, 2005* (2005).