Research Article

Performance Evaluation of Indices-Based Query Optimization from Flash-Based Data Centric Sensor Devices in Wireless Sensor Networks

Sanam Shahla Rizvi¹ and Tae Sun Chung²

¹ Department of Computer Sciences, Preston University, No. 85 Street 3, Sector H-8/1, Islamabad 44000, Pakistan
² School of Information and Computer Engineering, Ajou University, San 5 Woncheon-dong Yeongtong-gu, Suwon 443-749, Republic of Korea

Correspondence should be addressed to Sanam Shahla Rizvi, s.s.rizvi@preston.edu.pk

Received 30 July 2011; Revised 11 October 2011; Accepted 16 October 2011

Flash memory has become a more widespread storage medium for modern wireless devices because of its effective characteristics like nonvolatility, small size, light weight, fast access speed, shock resistance, high reliability, and low power consumption. Sensor nodes are highly resource constrained in terms of limited processing speed, runtime memory, persistent storage, communication bandwidth, and finite energy. Therefore, for wireless sensor networks supporting sense, store, merge, and send schemes, an energy efficient and reliable database-based query optimization technique is highly required with consideration of sensor node constraints. Databases on hard disk drives perform data storage and retrieval using index structures which are still not practiced for sensor devices. In this paper, we evaluate different indices like B-tree, R-tree, and MR-tree by implementing them on log structured external NAND flash memory-based advanced file systems for supporting energy efficient data storage and query optimization from flash based data centric sensor devices in wireless sensor networks. Experimental results show that PIYAS (Rizvi and Chung, 2010) file system along with B-tree indexing deployed on flash memory MLC gives the significant performance in respect of high query throughput optimization and less resources consumption for wireless sensor devices.

1. Introduction

The continuous improvement in hardware design and advances in wireless communication have enabled the deployment of various wireless applications. Wireless sensor network (WSN) applications become essential tools for monitoring the activity and evolution of our surrounding environment. The examples of WSN applications include environmental and habitat monitoring, seismic and structural monitoring, surveillance, target tracking, ecological observation, and a large number of other applications. In WSNs, monitoring can be deployed by the following three techniques. First, each sensor node transmits its generated data to a sink node immediately [1]. This approach is referred as Sense and Send. Second, every sensor node aggregates its own generated data and data coming from its children nodes and then sends them to its parent node [2]. This scheme is called Sense, Merge, and Send. Third, each sensor node stores its own generated data in its local memory. The data are aggregated and are sent to the sink node when they are queried [3]. This approach is called Sense, Store, Merge, and Send.

Currently, the advanced applications follow the third approach mentioned above. They store the sensor data in local on-chip and/or off-chip flash memory and perform in-network computation when required [4–6]. Such in-network storage approach significantly diminishes the energy and communication costs and prolongs the lifetime of sensor networks. As a result, many techniques in the areas of data centric storage, in-network aggregation, and query processing in WSNs have been proposed.

We compare previous schemes as shown in Table 1. Matchbox [7], the first file system for sensor nodes, provides only the append operation and does not allow the random
access of data for modification. It does not offer built-in features for device life efficiency in terms of wear-leveling. It has small size code and occupies reduced main memory footprints that rely on the number of open files. ELF [8] claims to outperform the Matchbox by higher read throughput and random access of data by timestamps. Like Matchbox and ELF, Capsule [9] is also a limited internal memory technique. It claims to outperform ELF in terms of energy efficiency. MicroHash [10] is an external large memory centric approach.

It appends the data in time series and uses the hash index structure for answering queries. It suffers from the need for extra I/O operations to maintain the huge metadata. However, none of the four previously discussed approaches consider the data life efficiency in terms of in-network data persistence as they simply erase the data to provide space for new data when the memory is exhausted. Storage efficiency in terms of optimal memory bandwidth utilization is also not guaranteed in the previous schemes as a small amount of data consume a complete memory page where remaining bytes remain unused. However, PIYA [11] and PIYAS [4] schemes provide long-term in-network data availability by retaining data in form of raw and aggregate data and provide optimal utilization of memory space by gathering data in main memory buffers. The data flush in the flash memory when the data of one complete page become available, except in exceptional cases where the sensor stops sensing and switches to its sleep mode. Plus, they offer high throughput with various natures of queries. Furthermore, PIYAS prolongs device life in terms of wear-leveling and offers higher energy efficiency.

Even though a recent study [12] shows that flash storage is two orders of magnitude cheaper than communication and comparable in cost to computation plus the fact that flash memory offers many other advantages in terms of large size and reliable storage, its special hardware read, write, and erase characteristics impose design challenges on storage systems [4, 13] (discussed in detail in Section 2.2). Additionally, due to the problems of flash memory, storage management techniques developed for disks may not be appropriate for flash.

Furthermore, for database systems, index structures are widely studied for hard disk as well as for flash memory [14]. However, in our knowledge, indices on flash based sensor devices in WSNs are never considered. Thus, we believe that implementation of indexing on resources-constrained sensor devices for data access comes up with significant performance. Therefore, to make flash media useful for sensor environments and to efficiently satisfy the network business goals and requirements relevant to sensor data storage and retrieval, a reliable data management scheme along with an efficient index structure is highly required.

In this paper, we experiment B-tree, R-tree, and MR-tree indices on advanced NAND flash-based memory management schemes to evaluate the performance effectiveness of query optimization for the resources-constrained sensor devices. Subject index structures are selected due to their sensor environment-oriented particular features, as B-tree is used for sequential data access, and it can be advantageous for networks accumulating data for environmental and habitat monitoring. R-tree and MR-tree structures are used for indexing multidimensional information like geographic information system. They can be significant to access the spatial objects such as restaurant locations and typical maps made for streets, buildings, outlines of lakes, and coastlines. Therefore, the increase in performance totally depends on selection of right type of index structure according to sensor environment.

The remainder of this paper is organized as follows. We review the background of system architecture of sensor node plus flash memory, and different indices characteristics are explained in Section 2. Comprehensive experiments are performed and discussed in Section 3. Finally, Section 4 presents the conclusions.

### 2. Background

#### 2.1. System Architecture of Sensor Node

The architecture of the wireless sensor node consists of a microcontroller unit (MCU) that interconnects a data transceiver, sensors along with analog-to-digital converters (ADCs), an energy source, and an external flash memory, see Figure 1.

The MCU includes a processor, a static RAM (SRAM), and on-chip flash memory. The processor increases efficiency by reducing power consumption. It runs at low frequency (~4–58 MHz) and further saves energy while the node is in standby or sleep mode. The low-power microcontrollers have limited storage, typically less than 10 KB of SRAM, mainly for code execution. However, in latest generation of sensors [5], it also uses for in-memory buffering. The limited amount of on-chip flash memory provides a small nonvolatile storage area (~32–512 KB). It is used for storing...
executable codes and accumulated values for a small period of time. However, it consumes most of the chip area and much of the power budget. Therefore, a larger amount of extra flash memory, perhaps more than a megabyte, is used on a separate chip to support the enhanced network functionality. The required amount of power can be obtained from many sources. Most sensors deploy a set of AA batteries and/or solar panels [15]. However, in most cases, the choice of correct energy source is application specific.

Flash memory is a nonvolatile solid state memory which has many attractive features such as small size, light weight, fast access speed, shock resistance, high reliability, and low-power consumption. Because of these attractive features, decreasing price, and increasing capacity, flash memory is becoming ideal storage media for mobile and wireless devices [16].

2.2. Overview of Flash Memory. Flash memory array is partitioned into equal size erase units called blocks, and each block is composed of a fixed number of read/write units called pages, see Figure 2. Every page has two sections: data area and spare area. Spare area stores metadata like logical block number (LBN), logical page number (LPN), erase count number (ECN), error correction code (ECC), cleaning flag for indicating garbage collection process in block, used/free flag to show that page is used or still free, and information of being valid/obsolete about data in data area. The sizes of pages and blocks differ by product.

Flash memory has three kinds of operations: page read, page write, and block erase. The performance of three kinds of operations is summarized based on memory access time and required energy at maximum values as shown in Table 2 [17].

| Operation          | Time (μsec) | Energy (mA) (Current 3.3 V) |
|--------------------|-------------|-----------------------------|
| Page read (512 + 16) B | 15          | 20                          |
| Page write (512 + 16) B | 500         | 25                          |
| Block erase (16 K + 512) B | 3,000      | 25                          |

Table 2: Performance of NAND flash memory.

| Operation          | Flash SLC | Flash MLC |
|--------------------|-----------|-----------|
| Capacity           | 64 GB     | 256 GB    |
| Reads              | 100 MB/s (max.) | 220 MB/s (max.) |
| Writes             | 80 MB/s (max.) | 200 MB/s (max.) |
| Endurance          | 100,000 cycles | 10,000 cycles |
| MTBF               | 2,000,000 hours | 1,000,000 hours |

Table 3: Specification comparison: SLC and MLC memory.

First Drawback. An inefficiency of in-place-update operation. When we modify data, we cannot update data directly at the same address due to the physical erase-before-write characteristics of flash memory. Therefore, updating even one byte data in any page requires an expensive erase operation on the corresponding block before the new data can be rewritten. To address this problem, the system software called flash translation layer (FTL) was introduced, as in [20–22]. FTL uses a non-in-place-update mechanism to avoid having to erase on every data update by using logical-to-physical address mapping table maintained in main memory. Under this mechanism, the FTL remaps each update request to different empty location and then the mapping table updates due to newly changed logical-to-physical addresses. This protects one block from being erased per overwrite. The obsolete data flagged as garbage which a software cleaning process later reclaims. This process is called garbage collection, as in [23–25].

Second Drawback. The number of erase operations allowed to each block is limited like 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 in [26, 27]. These drawbacks represent hurdles for developing a reliable flash memory-based sensor storage systems.

2.3. Index Structures. B-tree [28–30] is an optimized data structure that most representatively uses in hard disk drive-based database management systems and file systems.
It is an effective method to keep data sorted and perform data insert, delete, search, and sequential access efficiently in logarithmic amortized time. In B-tree, a node can have variable number of child nodes within some predefined range, as shown in Figure 3. In order to maintain the range, nodes may join or split. In B-tree, every parent/internal node keeps the keys/physical addresses of records rather than leaf nodes. It keeps the records in sorted order to sequentially traverse and uses a hierarchical index to minimize the number of memory units read to access a data record.

R-tree [31, 32] data structure is similar to B-tree, but it uses for spatial access methods like multimedia data, see Figure 4. Every node in R-tree can keep a predefined number of entries where each entry of leaf node consists of information about pointer to actual data element and depth of node. R-tree is height balancing structure that often increases the height of tree to balance the leaf nodes.

MR-tree [33, 34] data structure is similar to R-tree but the height between subtrees can be unbalanced with difference by maximum 1, as shown in Figure 5. When height crosses its threshold, it runs a height balance algorithm that rearranges the entries and splits the parent nodes. MR-tree nodes are classified in parent nodes, leaf nodes, and half leaf nodes where half leaf nodes refer to the node with 1 entry. By insertion of new objects in half leaf nodes, they become leaf nodes that have height of minimum 2.

3. Performance Evaluation

3.1. Simulation Methodology. To demonstrate the performance effectiveness of index structures in sensor environment, we performed a trace-based simulation. We applied different indices on PIYAS [4], PIYA [11], and MicroHash [10] schemes and analytically compared them on different parameters. Evaluation focuses on four parameters.

(i) Space Management. This shows the flash memory allocation against the thousands of continuous sensor readings and main memory consumption for maintaining the data buffers and metadata.

(ii) Search Performance. This shows the number of pages required to be read for responding to a query.

(iii) Throughput Performance. This shows the response of number of queries in a unit of time.

(iv) Energy Consumption. This shows the energy consumption while data writes to and data reads from sensor local flash memory.

We have built a simulator with 32 MB of flash space that is divided into erase blocks of equal size. Each block size is 16 KB, and every block is composed of 32 pages as read/write units. Every page size is 512 B with 16 B spare area. We extracted the trace file from COAGMET [35]. The two years raw data were extracted on an hourly basis from January 01, 2009 to December 31, 2010 from the Willington climate station. The trace file contains a total of 279,968 sensor readings, and it is a combination of all known data formats like negative, positive, and decimal values.

Every network has business rules to achieve some business goals. To achieve services in sensor networks, business rules are an effective method for programming a file system for sensor nodes. Rules are logically linked as chain where the structure of rules represents the simple business logic in a compact and efficient way. For example, the business goal says to collect the temperature readings in discrete range from 1F to 80F. In that case, we can split the range in set of rules like (A: [1–20]), (B: [21–40]), (C: [41–60]), and (D: [61–80]). The formulation of set of rules highly depends on the probability of type of data accumulation from environment and location for implementation of sensors. Since the sensor nodes assist the real life processes, the variation in set of rules is expected to address the monitoring of service parameters. Therefore, we assume that the set of rules is available to sensor nodes from network applications.
To prove the enhancement of our idea for large size of sensor data centric applications, we experimented with a broad range of rule values. Rules are adopted as directory buckets in case of MicroHash. The rules are given in Table 4.

The total elapsed time is calculated by (1) for effective comparison between schemes. Time required for read in unit of page from flash memory to data register is calculated by (2). Time required to read a byte unit from data register to main memory is calculated by (3). Time required for computation in main memory for building mapping structure and query processing framework is calculated by (4). Time required to write data from main memory to flash media is calculated by (5). For better understanding of experimental results in terms of time and energy, we refer to Table 2,

\[
\text{Total time} = T_{FR} + T_{RR} + T_{\alpha} + T_{WR},
\]

where

\[
T_{FR} = (\text{read count (page)} \times \text{read time}),
\]

\[
T_{RR} = (\text{read count (byte)} \times \text{read time}),
\]

\[
T_{\alpha} = \text{Time for Computation in RAM},
\]

\[
T_{WR} = (\text{write count (page)} \times \text{write time}).
\]

3.2 Experimental Results. Figure 6 shows the consumption of flash memory in number of erase blocks for number of sensor readings attempted by every rule. Trigger with every individual rule buffer (TgRule) is used in SRAM for sampling sensor readings. We show the fine granularity of data arrival in buffer of every rule by taking a small value of threshold as

| Rule symbol | Rule range (temperature) |
|-------------|--------------------------|
| A           | -99–0                    |
| B           | 1–100                    |
| C           | 101–200                  |
| D           | 201–300                  |
| E           | 301–400                  |
| F           | 401–500                  |
| G           | 501–600                  |

TgRule = 3 for PIYA and PIYAS schemes, and as MicroHash does not sample data, so we show the consumption of media for MicroHash by keeping trigger unset as TgRule = 0. In figure, flash blocks are individually allocated as chains to every rule for saving the sensor data corresponding to a trigger threshold where thousands of readings are stored in a very small flash memory space by both PIYA and PIYAS schemes. MicroHash stores data in linear sequential order. Therefore, we calculated blocks consumed by MicroHash by counting the number of pages allotted to every bucket. In this result, we only show the space consumed by data pages, and space assigned to metadata is not added. However, results clearly show the effectiveness of our memory management schemes. Our proposed schemes outperform the MicroHash for efficient media utilization.

Figure 7 shows the consumption of SRAM space in KB units while sensor filters and buffers the accumulated readings. SRAM provides opportunity to reserve data buffers
to put together the currently accumulated sensor readings from environment and then data store in a sensor’s local memory. Data buffering saves the flash space and reduces the write overhead. We reserve data buffers by the number of business rules where every buffer size is of one read/write unit of flash memory. When data arrives in the range of any rule, main memory space is assigned dynamically in chunks of bytes as buffer. Data of a complete buffer flush in flash memory when it becomes full. Results show that PIYAS scheme clearly outperforms both PIYA and MicroHash schemes. This is because unlike PIYA and MicroHash, PIYAS does not allocate static buffers, but buffers are allotted dynamically in chunks of bytes whenever some sensor reading arrives in the data buffer of some rule. Therefore, even though in a very write intensive scenario, PIYAS optimizes main memory space accumulation by 71.4% and 79.2% more than PIYA and MicroHash, respectively.

Flash memory mapping information stores in flash media in dedicated map blocks for fast initialization of system. At the time of system startup, mapping information fetches in SRAM. Limited SRAM and lengthy initialization time are challenging constraints of sensor resources. Therefore, to achieve instant mounting using very small size of SRAM footprints, data is saved sequentially on first available page of latest allocated block according to some rule. Where, every rule keeps only first available physical page number (PPN) in SRAM where single page mapping reserves only 2 B in main memory for 32 MB of flash memory which has $2^{16}$ total number of pages. Therefore, we need only the limited number of pages mapped by the number of rules.

At system initialization time, for building the mapping table, we extract mapping information from map blocks to the main memory. We obtain a fast mounting in 136.75 μs; it consumes 0.396 J and 154 B in SRAM. Both PIYA and PIYAS schemes use same time and number of bytes while mounting the mapping structure in main memory and for saving the mapping information back to the map blocks.

System fetches the metadata from map blocks and builds the query processing framework in main memory for entertaining the read intensive scenarios efficiently. PIYA scheme extracts timestamps by reading spare area of first page of every block and sets the time between two consecutive data blocks of same rule chain. Then, the table arranges in main memory for fast access of data. When some query comes in the range of some rule, system forwards that in corresponding block according to the desired time range of query. System evaluates the timestamp written in spare area from latest written page. If page supports queried value, then system checks the data items inside data section of page, otherwise it moves on page up.

In case of a large size of space being occupied, scanning by PIYA of spare area of first page of every block to build the mapping table and then finding the exact pages by reading spare areas of every page in the corresponding block consumes a long time and high energy. Therefore, PIYAS scheme implements a more energy efficient data access and provides a high throughput for responses to user queries. It maintains the data storage log in form of metadata in dedicated map blocks separated from the file system mapping information. It stores the metadata regarding memory assigned to every rule in a particular time interval. Though, in a read intensive scenario, PIYAS scheme preserves the average of 24 times more resources while building B-tree, R-tree, and MR-tree structures; it reserves average of 7.56 and 3.57 times more resources in terms of space, time, and energy while building the query processing framework in SRAM, compared to MicroHash and PIYA, respectively.

Figure 8 shows the query throughput by the average number of queries responded to per minute time unit. Evaluation is performed without applying any index structure on any scheme using their original mapping structures. Results
show that PIYAS greatly outperforms the previous PIYA and MicroHash schemes in time required for query responses with 90% and 84% queries per minute, respectively.

Figures 9, 10, and 11 present the results of query throughput by the average number of queries responded to per minute time unit which are obtained by applying B-tree, R-tree, and MR-tree indexes, respectively, on all three memory management schemes. Comparison between indices proves the dominance of B-tree with 18.2% better performance compared to R-tree which in turn improves 9% more throughput compared to MR-tree. However, the significance of index structures totally depends on selection of type of index according to sensor environment. Since, we experiment the traces obtained from environment and habitat monitoring sensors where the queries usually perform to store and access the data sequentially. Therefore, B-tree prominent here though R-tree and MR-tree has worth in spatial sensor environment.

Alternatively, results show that PIYAS scheme significantly outperforms PIYA with 6.57 times more throughput.
which in turn has advantage over MicroHash by 1.64 times more throughput. Lower performance of MicroHash is observed which can be because when flash space becomes exhausted and there is no space remaining for further data storage, scheme selects the victim block for garbage collection, and data in the victim block are simply erased and then the future queries cannot access such data. This generates a data failure for user applications.

However, PIYA and PIYAS congregate the values of victim block from flash erase unit to a read/write unit where every erase unit is composed of multiple read/write units. It means that the number of pages of victim block aggregate based on user-defined parameters like MIN, MAX, AVERAGE, COUNT, and so forth, on single page size. Therefore, every page on the aggregate data block represents the major information of data of one complete previously erased raw data block. This way, they preserve in-network data sustainability and availability by allocating aggregate data blocks to rules.

Further more, PIYAS enhances the scheme, conserves the energy, and takes a reduced search time for answering any query by allocating separate aggregate data blocks to individual rules to seek the exact data corresponding to particular rule values by avoiding the unnecessary read operation as PIYA does.

Table 5 shows the resources accumulation by the schemes addressed here. This information is calculated by obtaining the results of average number of pages that system reads on every request from network applications while searching the queried data in a very read intensive environment. Experimental results show that PIYAS optimizes 94.7%, 94.4%, 78.3%, and 66.7% resources in terms of time and energy compared to MicroHash, PIYA, R-tree, and MR-tree, respectively. However, it takes 20% more resources compared to B-tree.

![Table 5: Resources (time and energy) accumulation.](image)

| Scheme                  | Time (μs) | Energy (J) |
|-------------------------|-----------|------------|
| MicroHash               | 2884.2    | 3.135      |
| PIYA                    | 2702.04   | 2.937      |
| PIYAS                   | 151.8     | 0.165      |
| PIYAS (B-tree)          | 121.44    | 0.132      |
| PIYAS (R-tree)          | 698.28    | 0.759      |
| PIYAS (MR-tree)         | 455.4     | 0.495      |

Until now, almost all results show the performance dominance of PIYAS over other two previous schemes. It is also observed that PIYAS scheme along with B-tree index structure performs notably well. Therefore, to show the effectiveness of flash SLC [18] and MLC [19] devices in sensor environment, an experiment is performed for average read speed and read burst speed where burst throughput is the speed that data can be accessed from drive’s read-ahead memory register. This measures the speed of drive and controller interface. Such evaluation is performed using PIYAS scheme along with B-tree index structure.

Figure 12 demonstrates the results achieved per second in unit of time where MLC flash delivers by far the highest sustained more volume 48.59% for average sequential read and 57.3% for read burst throughput compared to SLC flash. Both flash SLC and MLC devices optimize random access data throughput with 0.1 ms.

4. Conclusion

This research evaluates the performance effectiveness of index structures to acquire the queried data from wireless sensor networks. Different indices like B-tree, R-tree, and MR-tree are compared by implementing them on advanced log-structured external NAND flash memory-based data management schemes called PIYAS, PIYA, and MicroHash. We performed trace-driven simulations to explore in detail also the effectiveness of SLC and MLC flash devices in sensor environment. Our comprehensive experimental results with real traces from environmental and habitat monitoring show that the B-tree index structure along with PIYAS memory management scheme comes up with significant performance in terms of time, energy, and space preservation.

Plus, we achieved instant mounting and reduced SRAM footprints by keeping a very low-mapping information size. The main memory required for accumulation of sensor readings is minimized. Storage utilization is optimized by effective data buffering in main memory before writing data to flash media. Data failure is mitigated by long-term in-network data availability. Fast access of memory to write data, computation in situ, high query throughput, more energy efficiency, and minimized reads, writes, and erases are effectively achieved.

Acknowledgments

The authors wish to thank Mr. M. A. S. Rizvi, Professor and Dean, Greenwich University, Karachi, Pakistan, and Mr. W. A. S. Rizvi, Plan Controller, Huawei Technologies Pakistan (Pvt) Ltd. for their valuable time for reviewing the whole manuscript and responding with their helpful comments.
This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0013487).

References

[1] J. Considine, F. Li, G. Kollias, and J. Byers, “Approximate aggregation techniques for sensor databases,” in Proceedings of the 20th International Conference on Data Engineering (ICDE ’04), pp. 449–460, Boston, Mass, USA, April 2004.

[2] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, “TAG: a tiny aggregation service for Ad-Hoc sensor networks,” in Proceedings of the Symposium on Operating Systems Design and Implementation, pp. 131–146, Boston, Mass, USA, 2002.

[3] D. Zeinalipour-Yazti, S. Neema, V. Kalogeraki, D. Gunopulos, and W. Najjar, “Data acquisition in sensor networks with large memories,” in Proceedings of the International Workshop on Biomedical Data Engineering (BMDE ’05), pp. 1188–1192, Tokyo, Japan, 2005.

[4] S. S. Rizvi and T. S. Chung, “PIYAS—Proceeding to intelligent service oriented memory allocation for flash based data centric sensor devices in wireless sensor networks,” Sensors, vol. 10, no. 1, pp. 292–312, 2010.

[5] D. Lymberopoulos and A. Savvides, “XYZ: a motion-enabled, power aware sensor node platform for distributed sensor network applications,” in Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN 2005, pp. 449–454, Los Angeles, Calif, USA, April 2005.

[6] A. Banerjee, A. Mitra, W. Najjar, D. Zeinalipour-Yazti, V. Kalogerakil, and D. Gunopulos, “RISE—Co-S: high performance sensor storage and co-processing architecture,” in Proceedings of the 2nd Annual IEEE Communications Society Conference on Sensor and AdHoc Communications and Networks (SECON ’05), vol. 2005, pp. 1–12, Santa Clara, Calif, USA, May 2005.

[7] D. Gay, “Design of matchbox: the simple filing system for motes,” TinyOS 1.x Distribution, 2003, http://www.tinyos.net.

[8] H. Dai, M. Neufeld, and R. Han, “ELF: an efficient log-structured flash file system for micro sensor nodes,” in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 176–187, Baltimore, Md, USA, November 2004.

[9] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, “Capsule: an energy-optimized object storage system for memory-constrained sensor devices,” in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, pp. 195–208, Boulder, Colo, USA, November 2006.

[10] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, D. Gunopulos, and W. A. Najjar, “An efficient index structure for flash-based sensor devices,” in Proceedings of the USENIX Conference on File and Storage Technology, pp. 31–44, San Francisco, Calif, USA, 2005.

[11] S. S. Rizvi and T. S. Chung, “PIYAS—Proceeding to Intelligent service oriented memory Allocation for flash based sensor devices in wireless sensor networks,” in Proceedings of the 3rd International Conference on Convergence and Hybrid Information Technology (ICHIT ’08), pp. 625–630, Busan, Korea, November 2008.

[12] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy, “Ultra-low power data storage for sensor networks,” in Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN ’06), pp. 374–381, Nashville, Tenn, USA, April 2006.

[13] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” ACM Computing Surveys, vol. 37, no. 2, pp. 138–163, 2005.

[14] H. S. Lee, H. Y. Song, and K. C. Kim, “Performance of Index trees on flash memory,” in Proceedings of the International Conference in Principles of Information Technology and Applications, pp. 725–734, 2007.

[15] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, “Hardware design experiences in ZebraNet,” in Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 227–238, Baltimore, Md, USA, November 2004.

[16] B. Dipert and M. Levy, Designing with Flash Memory, Anna-books Publisher, Poway, Calif, USA, 1993.

[17] Samsung Electronics NAND Flash Memory, K9F5608U0D data book, Samsung Electronics, Jung-gu Seoul, South Korea, 2011.

[18] Samsung Electronics NAND Flash Memory, MCCOE64G5-MPP-0VA Data Book, Samsung Electronics, Jung-gu Seoul, South Korea, 2011.

[19] Samsung Electronics NAND Flash Memory, MMD0E56G5-MXP-0VB Data Book, Samsung Electronics, Jung-gu Seoul, South Korea, 2011.

[20] T. S. Chung and H. S. Park, “STAFF: a flash driver algorithm minimizing block erasures,” Journal of Systems Architecture, vol. 53, no. 12, pp. 889–901, 2007.

[21] T. S. Chung, D. J. Park, S. Park, D. H. Lee, S. W. Lee, and H. J. Song, “A survey of Flash Translation Layer,” Journal of Systems Architecture, vol. 55, no. 5–6, pp. 332–343, 2009.

[22] S. J. Kwon and T. S. Chung, “An efficient and advanced space-management technique for flash memory using reallocation blocks,” IEEE Transactions on Consumer Electronics, vol. 54, no. 2, pp. 631–638, 2008.

[23] T. S. Chung, M. Lee, Y. Ryu, and K. Lee, “PORCE: an efficient power off recovery scheme for flash memory,” Journal of Systems Architecture, vol. 54, no. 10, pp. 935–943, 2008.

[24] L.-Z. Han, Y. Ryu, T.-S. Chung, M. Lee, and S. Hong, “An intelligent garbage collection algorithm for flash memory storages,” in Proceedings of the Computer Science and Its Applications, Lecture Notes in Computer Science, pp. 1019–1027, Glasgow, UK, 2006.

[25] L. Han, Y. Ryu, and K. Yim, “CATA: a garbage collection scheme for flash memory file systems,” in Proceedings of the Ubiquitous Intelligence and Computing, Lecture Notes in Computer Science, pp. 103–112, Wuhan, China, 2006.

[26] L. P. Chang, “On efficient wear leveling for large-scale flash-memory storage systems,” in Proceedings of the ACM Symposium on Applied Computing, pp. 1126–1130, Seoul, Korea, March 2007.

[27] Y. H. Chang, J. W. Hsieh, and T. W. Kuo, “Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design,” in Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC ’07), pp. 212–217, San Diego, Calif, USA, June 2007.

[28] B. C. Ooi and K. L. Tan, “B-trees: bearing fruits of all kinds,” in Proceedings of the 13th Australasian Database Conference, Melbourne, Australia, 2002.

[29] J. H. Nam and D. Park, “The efficient design and implementation of the B-tree on flash memory,” in Proceedings of the 23rd Korea Information Science Society, pp. 5–7, Seoul, Korea, 2005.

[30] C. H. Wu, L. P. Chang, and T. W. Kuo, “An efficient B-tree layer for flash memory storage systems,” in Proceedings of the
A. Guttman, “R-trees: a dynamic index structure for spatial searching,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 47–57, 1984.

C. H. Wu, L. P. Chang, and T. W. Kuo, “An efficient R-tree implementation over flash-memory storage systems,” in Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pp. 17–24, New Orleans, La, USA, November 2003.

K.-C. Kim and S.-W. Yun, “MR-tree: a cache-conscious main memory spatial index structure for mobile GIS,” in Proceedings of the Web and wireless geographic information systems, pp. 167–180, LNCS, 2005.

H. S. Lee, H. Y. Song, and K. C. Kim, “Performance improvement of MR-Tree operations on NAND Flash memory,” International Journal of Computer Science and Information Systems, vol. 3, no. 1, pp. 59–70, 2007.

“COAGMET,” 2011, http://ccc.atmos.colostate.edu/~coagmet/index.php.