Poster Abstract: TINX – A Tiny Index Design for Flash Memory on Wireless Sensor Devices

Ajay Mani Manjunath Rajashekhar Philip Levis
Stanford University, Department of Computer Science
{ajaym, manj, pal}@cs.stanford.edu

Categories and Subject Descriptors: H.3.1 [**Information Storage and Retrieval**]: Indexing methods.

General Terms: Algorithms, Design, Experimentation.

Keywords: Sensor Networks, Storage, Flash memory, Wear-leveling, Energy Efficiency, Indexing.

1. MOTIVATION

Flash memory is a cheap, viable storage alternative for the low power, energy constrained sensor nodes. It is still not clear however, what storage abstractions are best suited to Sensornet applications. Many authors have proposed a file-system based approach [1, 5] to storage on sensor devices. File systems, however, were originally designed to be a flexible and user-friendly naming interface to storage. In contrast, sensor applications often know their storage requirements in advance and do not have a human operator.

The primary purpose of sensor networks is to sense and process readings from the environment. Given this usage model, once a sensor node acquires a reading, it could transmit out the readings to the base-station immediately, or it could log it onto a flash device for later processing. For many application scenarios, [6] shows that logging data into flash memory is much cheaper than transmitting it over the communication network. In other situations, in-network query processing performance could be enhanced by locally archiving, processing and indexing. Sensor devices thus need an efficient way to retrieve data from storage in order to satisfy queries.

The peculiar read, write and erase characteristics of flash memory [2], imply that index data structures and other storage management techniques developed for disks, which often depend on in-place modification, may not be appropriate for flash. This motivates us to explore sophisticated data structures and algorithms that work around the constraints and limitations of flash memory to provide an efficient indexing mechanism.

2. PROBLEM STATEMENT

A sensor node is capable of sensing many different attributes that are either periodically sampled or event-driven data. We would like to build an index over the logged sensor readings which would support *value-based range queries*, t*ime-based range queries*, *hybrid queries* (combination of value-based and time-based range queries with && and || operators) and a*ggregation queries* (like COUNT, MIN, MAX and AVERAGE). The indexing scheme should also maximize wear-leveling, minimize erases and writes, minimize RAM structures and maintain the index and data on flash to facilitate easy block reclamation.

SenSys'06, November 1–3, 2006, Boulder, Colorado, USA.
ACM 1-59593-343-3/06/0011.

3. DESIGN

3.1 Organization

Segments: TINX divides flash memory into logical units of contiguous blocks called *Segments*. TINX uses segments as a circular log. Each segment is a self contained unit that contains data and the index structure over that set of data. Addressing within a segment is relative, and the segments are ordered in time. This logical division forms an automatic second level index. Because applications are typically interested in data or events that have occurred in recent history, it also ensures that searches are restricted to only a subset of segments and not over a monolithic index, hence reducing the number of pages read for satisfying a query. Reclaiming of data blocks also becomes easy, as erasing an entire segment will not lead to any reorganization or impact on the rest of the data or index.

Page: Segments are composed of two classes of pages: data and index. Pages are consumed in increasing addressing order in a segment, and there is no special partitioning of the segment demarked. This scheme helps ensures wear leveling of the pages in the flash.

Data Pages: The data page holds the data records. When a data page is full, a new data page is allocated, and is linked to the previous data page.

Index Pages: An index page consists of entries that are a key of the tuple and a pointer to the data record. The index page header contains pointers to other index pages, as well as the max and min values that this index page covers.

3.2 The Indexing Mechanism

TINX supports having multiple indexes. Each index page stores part of one index and is organized in a special tree fashion. An index page covers a range of values, and the two pointers in the header point to two index pages (children) that cover a finer range. Notice in Figure 1 that

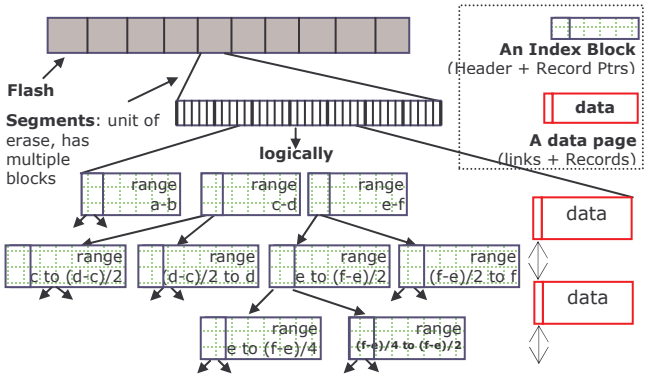

Figure 1 – Logical Organization of index and data blocks

the first level of the index cover a pre-divided range of values, and each of the two children cover half of the range of its parents. TINX maintains a secondary RAM index of the segment which it is currently writing to. This index is heavily used during the growing phase of the flash memory. We are investigating into adding this onto the flash hierarchy to help the search algorithm and reduce the burden on RAM.

3.3 Algorithm

Growth: TINX buffers sensed readings to a RAM page and flushes the data to flash only when the page is full. When it flushes data, TINX sorts its index values and appends references to the records in respective index pages. A cached second level index helps point to the appropriate index pages to retrieve for editing. All changes to the index pages are restricted to appends. When an index page is full, new children that correspond to the split ranges of their parents are created and the new data record is inserted into one of their children.

Search: Given a value to search for, TINX walks the index tree for that value for the relevant segment. At each pass in a segment, TINX generates a bitmap of the data pages that contain matches. It then looks up the records in these pages. This bitmapped manner of searching also helps in reducing the data pages read for hybrid queries; where bitmaps from different indexes are combined logically to retrieve pointers to data pages that match the query. As the records are already sorted on time in the chained data pages, time based searches are just a modification of binary search.

4. EVALUATION

We simulated TINX in C, and used the dataset from [3, 4] to test its performance and overhead.

Energy and Space Overhead: The space overhead due to index pages acts as an indicator of energy consumed in maintaining the index. In our simulation, for an 8-byte index and records greater than 40 bytes, space overhead was within 15% of the total record size.

Search Performance: An appropriate index reduces the number of data pages which have to be read to satisfy a search query. In our simulation, an equi-valued search reads at most 2500 pages for 0.6 million data records placed in 150,000 data pages.

5. RELATED WORK

As far as we know, MicroHash [7] is the only other paper to look into the problem of building index structures over flash devices keeping in mind sensor devices. TINX provides a richer feature set than MicroHash which does not address range queries over value based searches. TINX also supports complex queries. Other systems [8, 9] that build B-Tree index structures over flash use address translation tables and have very large memory footprints (order of 2-3MB).

6. CONCLUSION AND FUTURE WORK

TINX organizes data and index to reduce search latency and maximize wear-leveling. We have started the initial process of porting the code into TOSSIM and TinyOS and are building a storage board to explore NAND/NOR tradeoffs and explore multi-role flash based systems. We are looking forward to identify the various tradeoffs in data-size, index structures and schemes, and also come up with a user API to build a stable, usable and search optimized data store for sensor devices.

7. REFERENCES

[1] H. Dai, M. Neufeld, and R. Han, ELF: Efficient Log-Structured Flash File System for Wireless Micro Sensor Nodes. SenSys 2004

[2] E Gal, S Toledo: Algorithms and data structures for flash memories. ACM Comp. Surv. 37(2): 138-163 (05)

[3] http://berkeley.intel-research.net/labdata/

[4] http://robotics.usc.edu/~namos/data.html

[5] http://tinyos.net/tinyos-1.x/doc/matchbox-design.pdf

[6] G. Mathur, P. Desnoyers, D. Ganesan, P. J. Shenoy: Ultra-low power data storage for sensor networks. IPSN 2006: 374-381

[7] D. Zeinalipour-Yazti, S. Lin, V. Kalogeraki, W. A. Najjar, D. Gunopulos: MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices. FAST 05

[8] Wu C-H., Chang L-P., Kuo T-W., An Efficient BTree Layer for Flash Memory Storage Systems, In RTCSA, New Orleans, pp. 17-24, 2003.

[9] Wu C-H., Chang L-P., Kuo T-W., An Efficient Rtree Implementation Over Flash-Memory Storage, In RTCSA, Taiwan, pp. 409-430, 2003