

Efficient Indexing Data Structures for Flash-Based Sensor Devices

SONG LIN

University of California, Riverside

DEMETRIOS ZEINALIPOUR-YAZTI

University of Cyprus

and

VANA KALOGERAKI, DIMITRIOS GUNOPULOS, and WALID A. NAJJAR

University of California, Riverside

Presenter: Xiangnan Xu



Outline

- Introduction
- The Memory Hierarchy
- Problem Definition
- The Data Structures
- Indexing and Searching in MicroHash
- Indexing and Searching in MicroGF
- Experimental Methodology
- Experiment Evaluation
- Conclusion

Introduction

- Background

WSD : Wireless Sensor Devices , tiny computers on chips

Usage: Equipped with a low-frequency processor and a wireless radio, it can sense parameters of the physical world at extremely high resolutions, such as light, sound, temperature, humidity, pressure, etc.



Introduction

- Challenges

Storage and retrieval of sensor data

- Techniques

Centralized Repository

the acquisition of data from the physical world is succeeded by transmission of the respective data to the sink (querying node)

Situ Data Storage

keep a large window of measurements at the generating site, and transmit specific information to the sink only when requested

Introduction

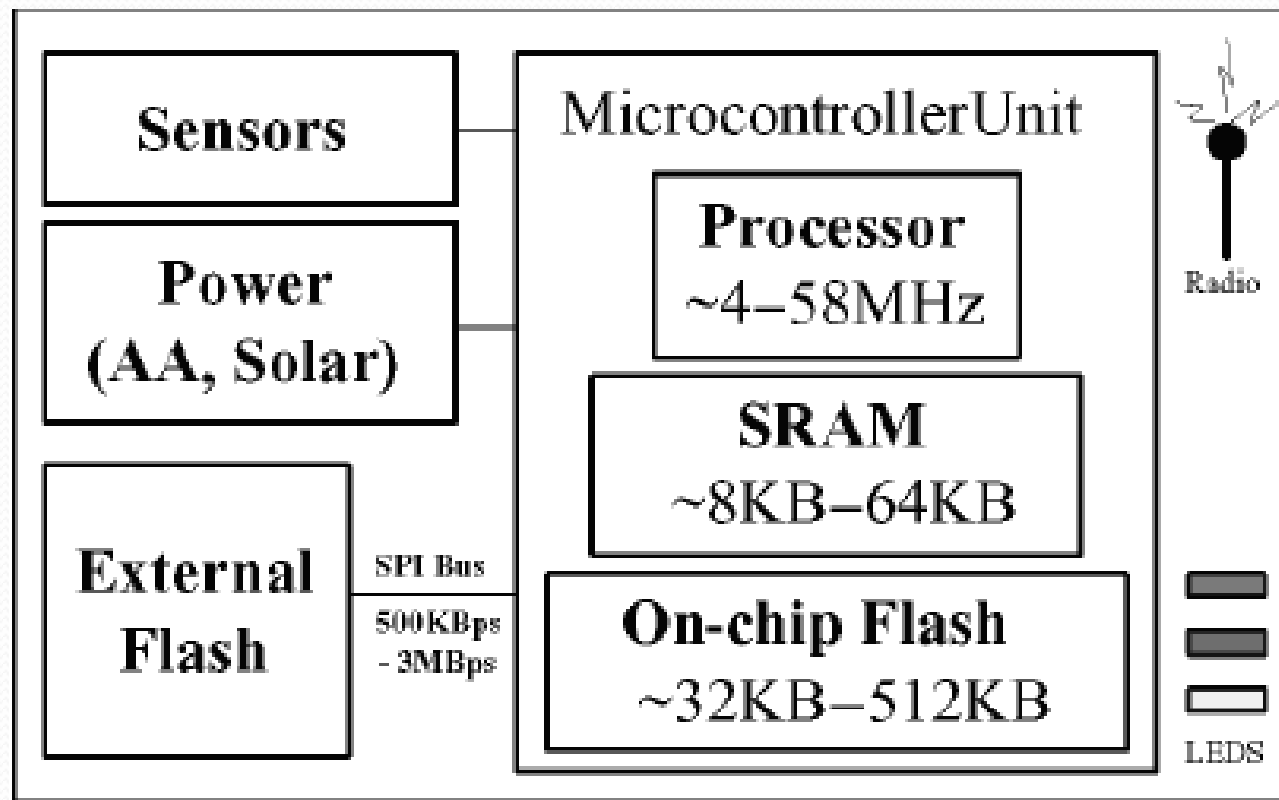
- Contributions
 - two indexing data structures, MicroHash and MicroGF
 - efficient algorithms for inserting, deleting, and searching data records stored on flash media
 - experimental study of the approach efficiency

MicroHash: support equality queries in sensor nodes with limited processing capabilities and a low energy budget

MicroGF: support spatial queries in sensor nodes equipped with GPS capabilities

The Memory Hierarchy

- System Architecture of Wireless Sensor



The Memory Hierarchy

- Flash Memory Type

NAND: new, faster erase time, higher durability and higher density, write at page granularity

NOR: old, faster access times, write at byte granularity

- Distinct Constraints of **NAND**

- Delete Constraint: at block granularity

- Write Constraint: at page granularity after the respective block was deleted

- Wear Constraint: each page can only be written a limited number of times

Problem Definition

- Value-Based Equality Queries (MicroHash, MicroGF)
1D query $Q(v_i, a)$
- Time-Based Range and Equality Queries (MicroHash, MicroGF)
 $Q(t, a, b)$
 $Q(t, a, b), a=b$
- Spatial Queries (MicroGF)
 $Q(v_1, v_2, \dots, v_x, A_{\text{query}})$, spatial attributes are in the query area A_{query}

The Data Structures

- In-Memory (SRAM) Data Structures

```
typedef struct Page {
    uint8_t  typ:3;
    uint16_t crc:16;
    uint16_t pwc:15;
    uint8_t  siz:7;
    uint32_t ppa:23;
    union {
        RootP rootP;
        DirP  dirP;
        IdxP  idxP;
        DataP dataP;};
} __attribute__((packed));

typedef struct DataP {
    DataRec records[DREC];
} __attribute__((packed));
```

```
typedef struct IdxP {
    // optional anchor
    uint64_t lastTS;
    IdxRec records[IREC];
} __attribute__((packed));

typedef struct DataRec {
    timestamp_t ts;
    data_t      val1;
} __attribute__((packed));

typedef struct IdxRec {
    fladdress_t datap;
    // optional offset
    floffset_t  offset;
} __attribute__((packed));
```



Indexing and Searching in MicroHash

- Indexing in MicroHash
 - Initialization phase
 - load the root page and certain parts of the directory into SRAM
 - Growing phase
 - data and index pages are sequentially inserted and organized on the flash media
 - Repartition phase
 - index directory is reorganized so that directory buckets with the highest hit ratio remain in the memory
 - Deletion phase
 - triggered for garbage collection purposes



Indexing and Searching in MicroHash

- Searching by value

A record-at-a-time query return mechanism in which records are reported to the caller on record-by-record basis.

- Searching by Timestamp

- LBSearch

Starts by setting a pessimistic lower bound on which page to examine next, and then recursively refines the lower bound until the requested page is found

- ScaleSearch (superior, uniformly distributes data and index pages on the flash media)

Exploits knowledge about the underlying distribution of data and index pages in order to offer a more aggressive search method that usually executes faster

Indexing and Searching in MicroHash

- Search Optimization

- **Elf-like chaining (ELC)**

copy the last nonfull index page into a newer page whenever new index records are added to the index until an index page becomes full

- **Two-phase read**

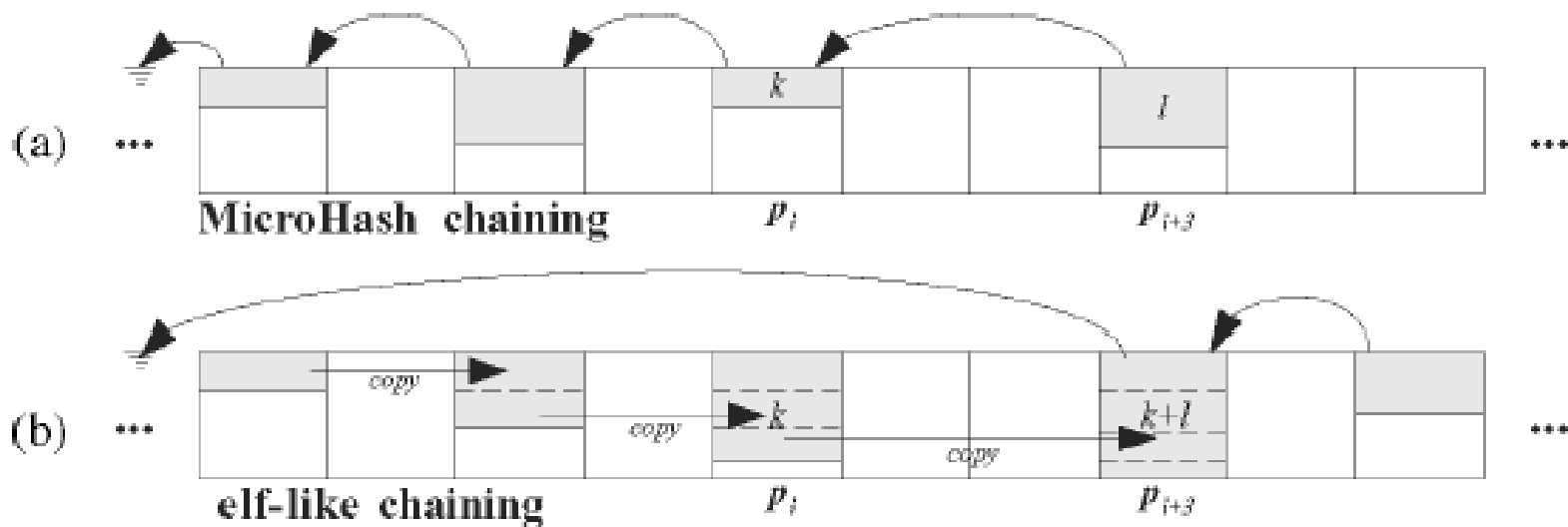
MCU reads a fixed header of a page from flash in the first place in the first phase, and then reads the exact amount of bytes in the next

- **Lossless Compression by exploiting Temporal Locality**

utilize run-length encoding scheme to eliminate the repetitive sequences which is a result of temporal locality

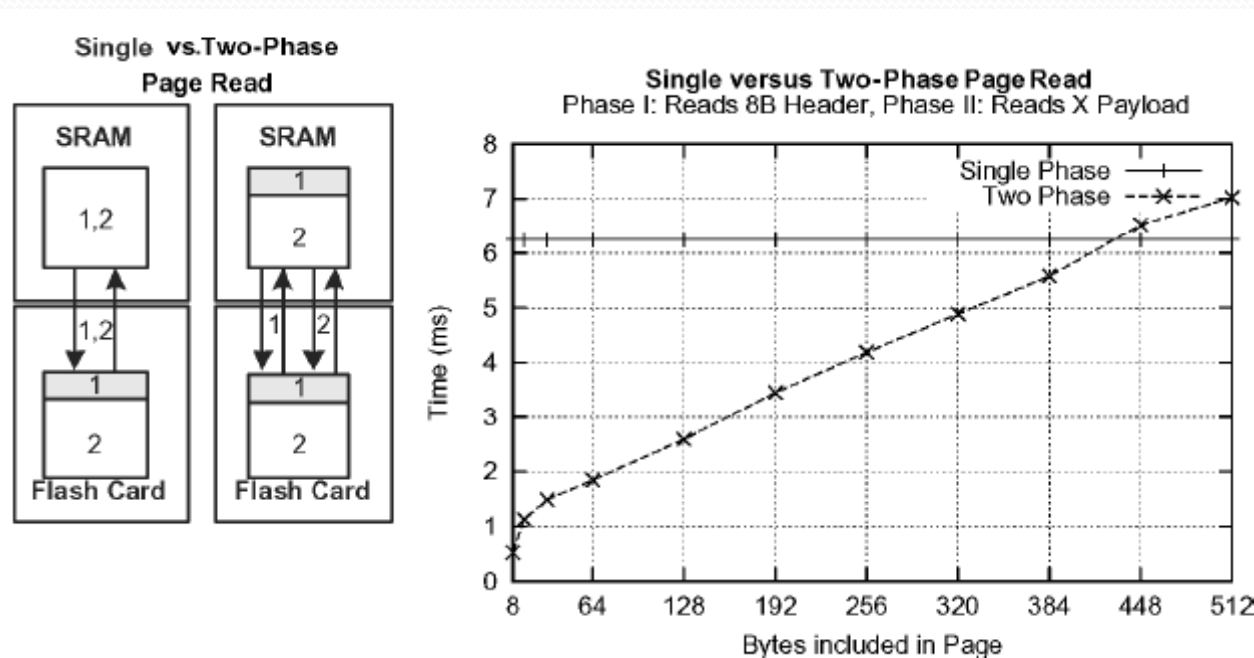
Indexing and Searching in MicroHash

- Index chaining methods



Indexing and Searching in MicroHash

- Two-Phase Page Reads



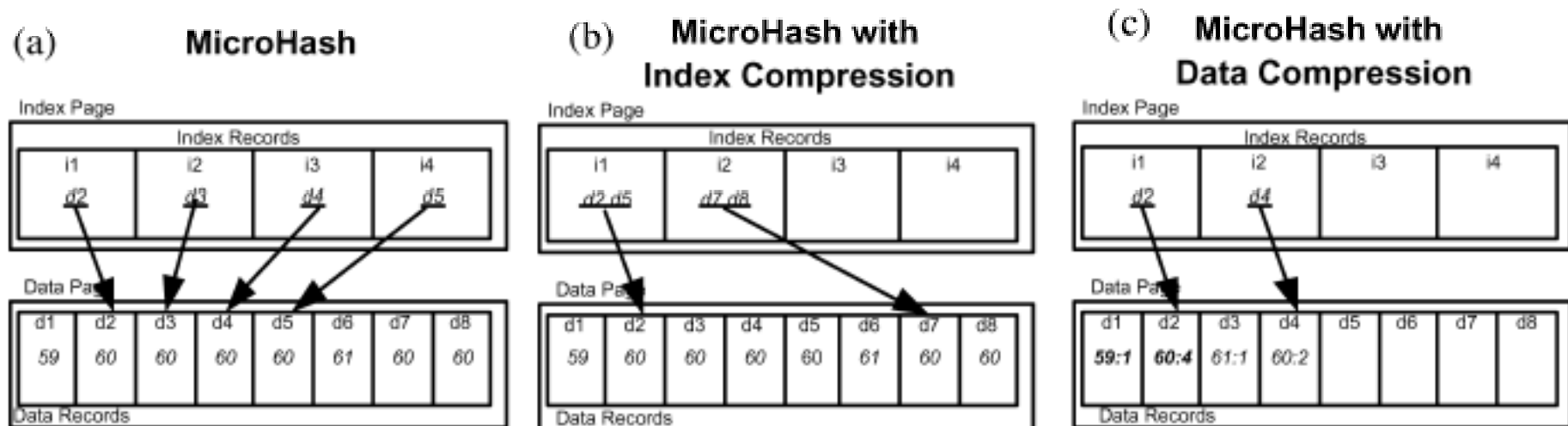
- Minimize the page reading time minimizes energy consumption during searches

Indexing and Searching in MicroHash

- Lossless Compression by Exploiting Temporal Locality

Run-length encoding scheme: turn consecutive value into a single value of the repeated value

e.g. $\{50, 60, 60, 60, 60, 60, 62, 62\} \rightarrow \{1:50, 5:60, 2:62\}$





Indexing and Searching in MicroGF

- MicroGF Index Data Structures

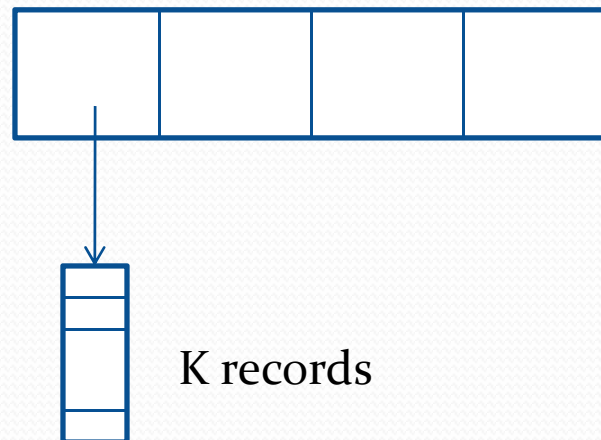
- Directory page data structure

the directory consists of an N-dimensional grid of cells, $n \times n$ square cells, n is the size of some geographic area, each cell contains the address of the last-known index page that mapped into this geographic region

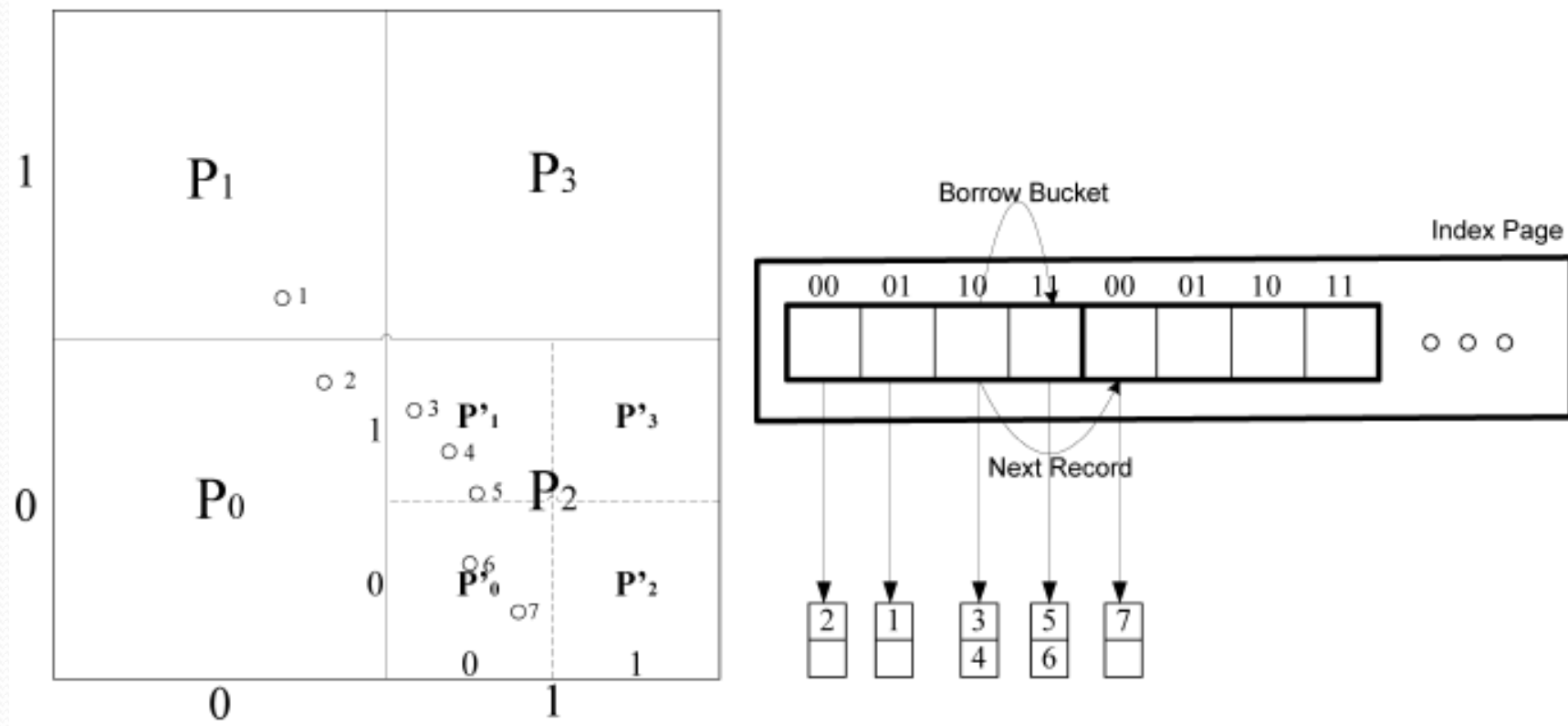
Indexing and Searching in MicroGF

➤ Index page data structure

The index pages contain the addresses of the respective data pages. Each index record is divided into four equal-size quadrants. Each quadrant maintains the address for K records which map to the specific area



Indexing and Searching in MicroGF





Indexing and Searching in MicroGF

- Comparison of methods to index spatial records
 - Grid Files
 - The directory in grid files is very large and cannot be maintained in SRAM
 - Quadtrees
 - poor index space utilization
 - MicroGF
 - better space utilization
 - query processing is faster and economical



Experimental Methodology

- Experimental Testbed
 - Implement MicroHash with a tiny LRU BufferManager in nesC, the programming language of TinyOS
 - Use one page as a write buffer, two pages for reading, one page as an indexing buffer, one for the directory, one final page for the root page
 - Wrote three libraries in order to run the code in TOSSIM, the simulation environment of TinyOS



Experimental Methodology

- PowerTOSSIM

A power modeling extension to TOSSIM, useful and accurate for modeling energy in a simulation environment

- Trace-driven experimental methodology

Traces come from two projects

- Great Duck Island (GDI 2002)

- INFATI

Experimental Evaluation

- Evaluation parameters

- **Space overhead**

overhead of maintaining the additional index pages, the overhead ratio is defined as follows:

$$\Phi = \frac{IndexPages}{DataPages + IndexPages}$$

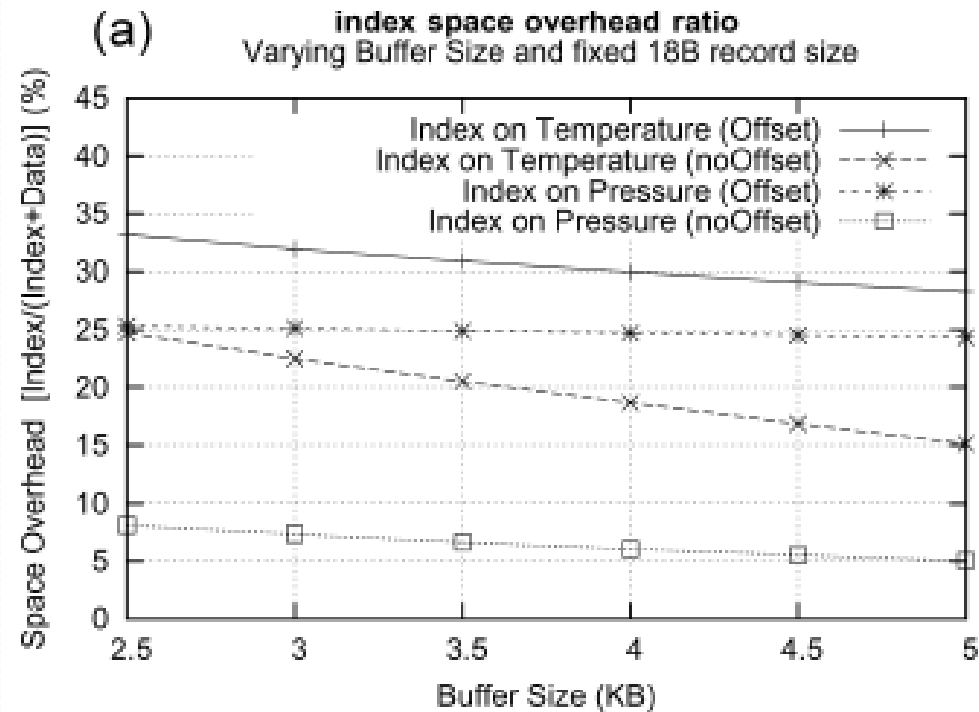
- **Search performance**

the average number of pages accessed for finding the required record

- **Energy consumption** for indexing the data records

Experimental Evaluation

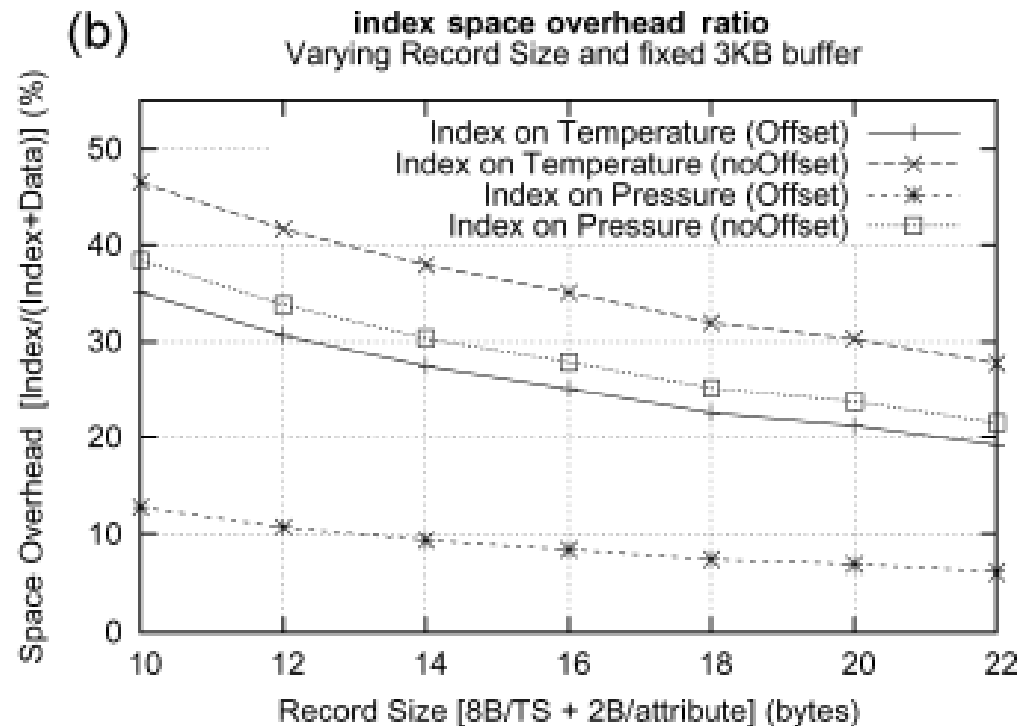
- Space Overhead--Increasing Buffer Size



- A larger buffer reduces overall space overhead
- NoOffset index record layout reduces the space overhead

Experimental Evaluation

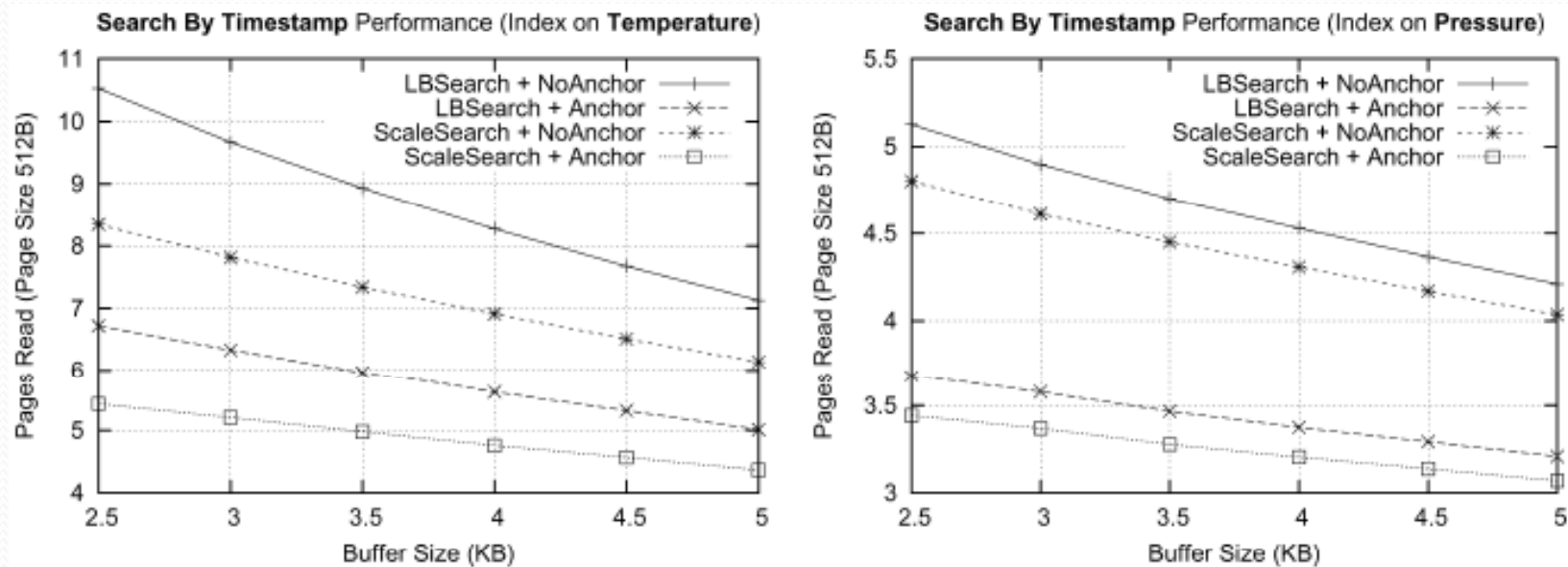
- Space Overhead--Increasing Data Record Size



- a larger data record size decreases the space overhead proportion

Experimental Evaluation

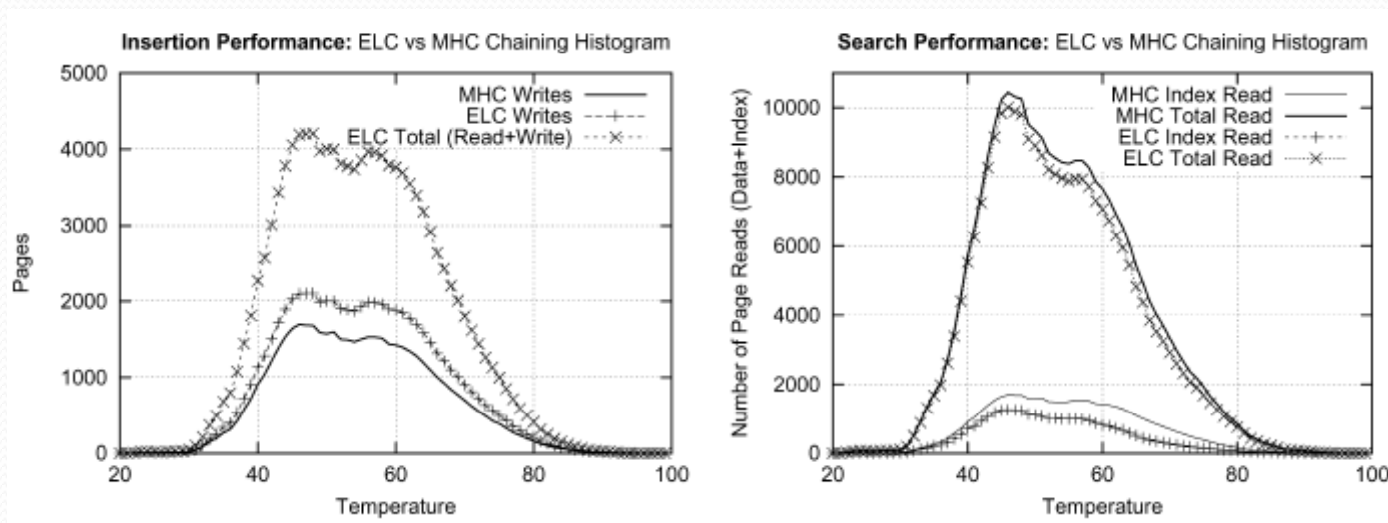
- Searching Performance– searching by timestamp



- Using an anchor inside an index page is a good choice
- ScaleSearch is superior to LBSearch
- ScaleSearch is able to find a record by its timestamp in approximately 3.5-5 page reads

Experimental Evaluation

- Searching Performance– searching by value



- MHC always requires less page writes than ELC
- ELC requires as many page reads as page writes in order to index all records
- Reading of data pages dominates the overall reading cost
- ELC only reduces the overall read gain by about 10%

Experimental Evaluation

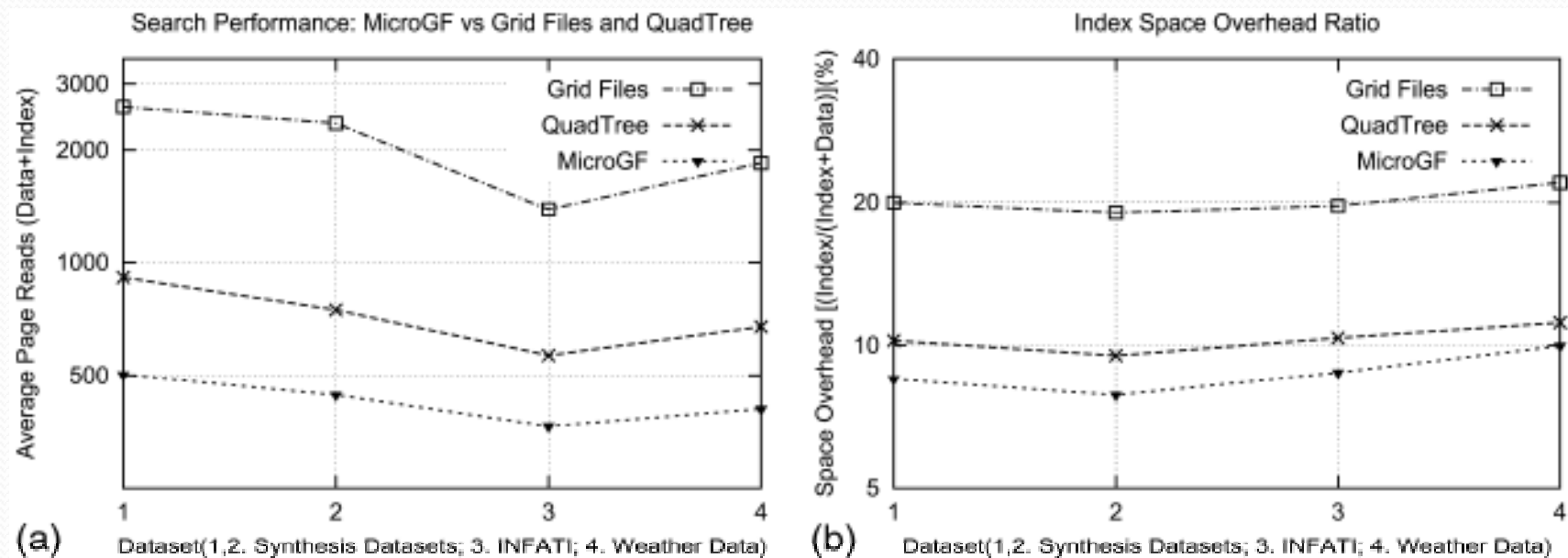
Table II. Performance Results from Indexing and Searching the Great Duck Island Dataset

Index On Attribute	Overhead Ratio Φ (%)	Energy Index (mJ)	ScaleSearch Avg Page Read
Light	26.47	4,134	4.45
Temperature	27.14	4,172	5.45
Thermopile	24.08	4,005	6.29
Thermistor	14.43	3,554	5.10
Humidity	7.604	3,292	2.97
Voltage	20.27	3,771	4.21

- Index pages never require more than 30% more space on the flash media
- Indexing the records has only a small increase in energy demand
- We can find any record by its timestamp with 4.75 page reads on average

Experimental Evaluation

- Comparison among MicroGF, Grid Files and Quadtree



- The MicroGF algorithm accesses 40% less pages than quadtree and 80% less pages than grid files in query processing
- The index space overhead of MicroGF is 15% less than quadtree and 56% less than grid files



Conclusion

- The experiment shows that the structures MicroHash and MicroGF are both efficient and practical
- The proposed access methods might provide a powerful new framework to realize in situ data storage in sensor networks
- We expect that these index structures will enable new types of queries that have not been addressed adequately to-date, such as temporal or top-k queries



Thank you!