Lightweight Indexing of Observational Data in Log-Structured Storage

Sheng Wang, David Maier, Beng Chin Ooi

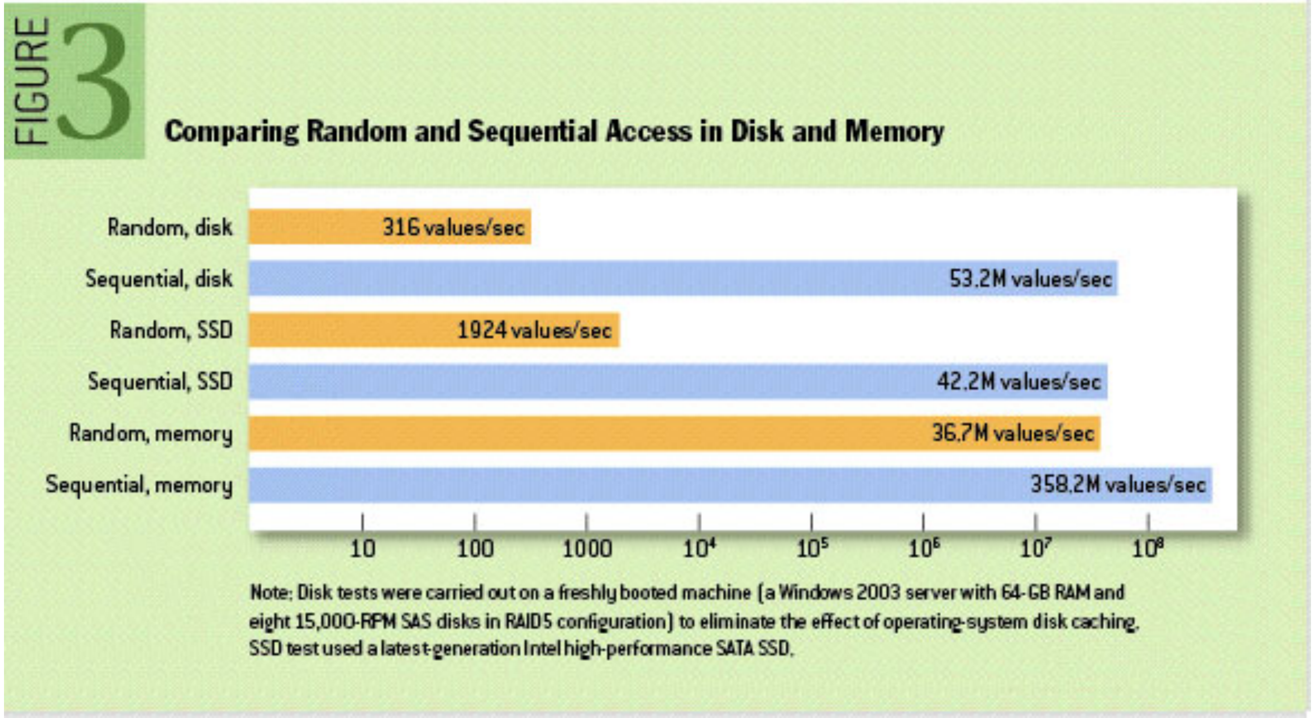


FIGURE 3

Comparing Random and Sequential Access in Disk and Memory

Note: Disk tests were carried out on a freshly booted machine (a Windows 2003 server with 64-GB RAM and eight 15,000-RPM SAS disks in RAID5 configuration) to eliminate the effect of operating-system disk caching. SSD test used a latest-generation Intel high-performance SATA SSD.

Preliminaries

Image taken from ACM

Observational Data

- Data collected from sensors.
- Following properties:
    - **Velocity:** The rate of data ingestion is very high. (**write-intensive)**
    - **Immutable:** Inherently the data is never changed after storing.
    - **Continuity:** Most sensors measure a continuous variable like temperature.
- We are interested in **time-range queries** (time series analysis) and **value-range queries** (anomaly detection)

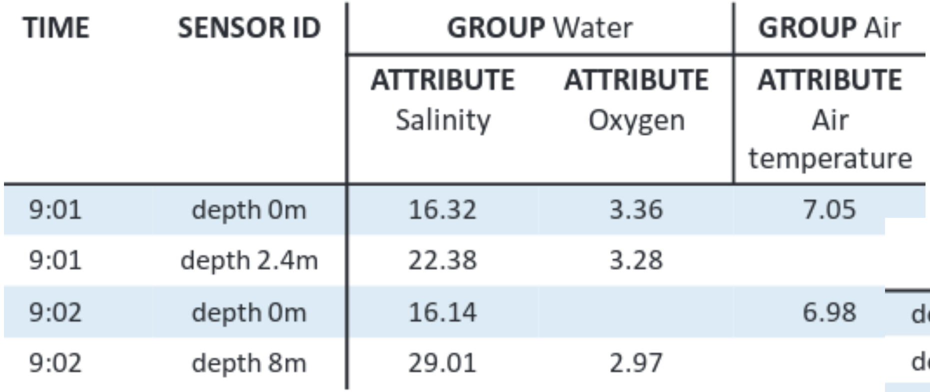
Data locality

- **Temporal locality:** The insertion time into the database correlates to observation time
- **Spatial Locality:** Sensors nearby each other will have similar values
- **Continuity:**
  - **Continuous variable:** Functions attains all values between end points.
  - **Continuous measurement:** maximum change after each step is bounded.

Log Structured Storage

- Records are appended to a file in insert order. (Excellent write performance and no random IOs like B+-tree for index maintenance)
- Two types:
  - **Ordered**: Keep records in RAM and sort before flushing to disk.
  - **Unordered**: Directly append to file.
- Authors use *LogBase* which is an open source unordered log-store.
  - Records are broken up into attributes and grouped if needed.

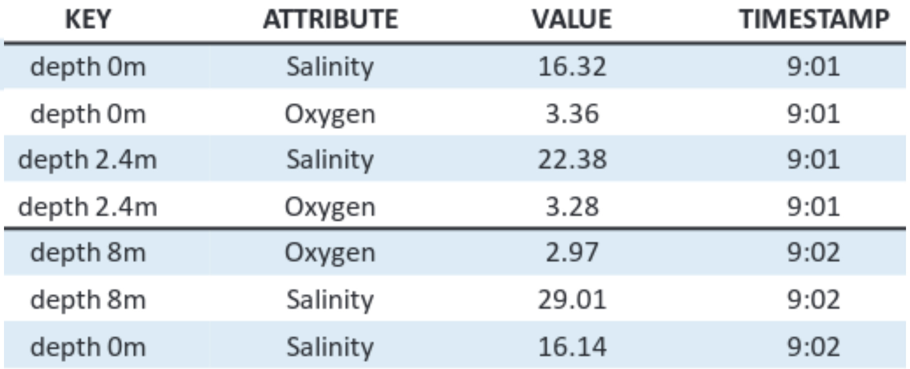
LogBase



TIME	SENSOR ID	GROUP Water		GROUP Air
		ATTRIBUTE Salinity	ATTRIBUTE Oxygen	ATTRIBUTE Air temperature
9:01	depth 0m	16.32	3.36	7.05
9:01	depth 2.4m	22.38	3.28	
9:02	depth 0m	16.14		6.98
9:02	depth 8m	29.01	2.97	

Figure 2: Schema logical view

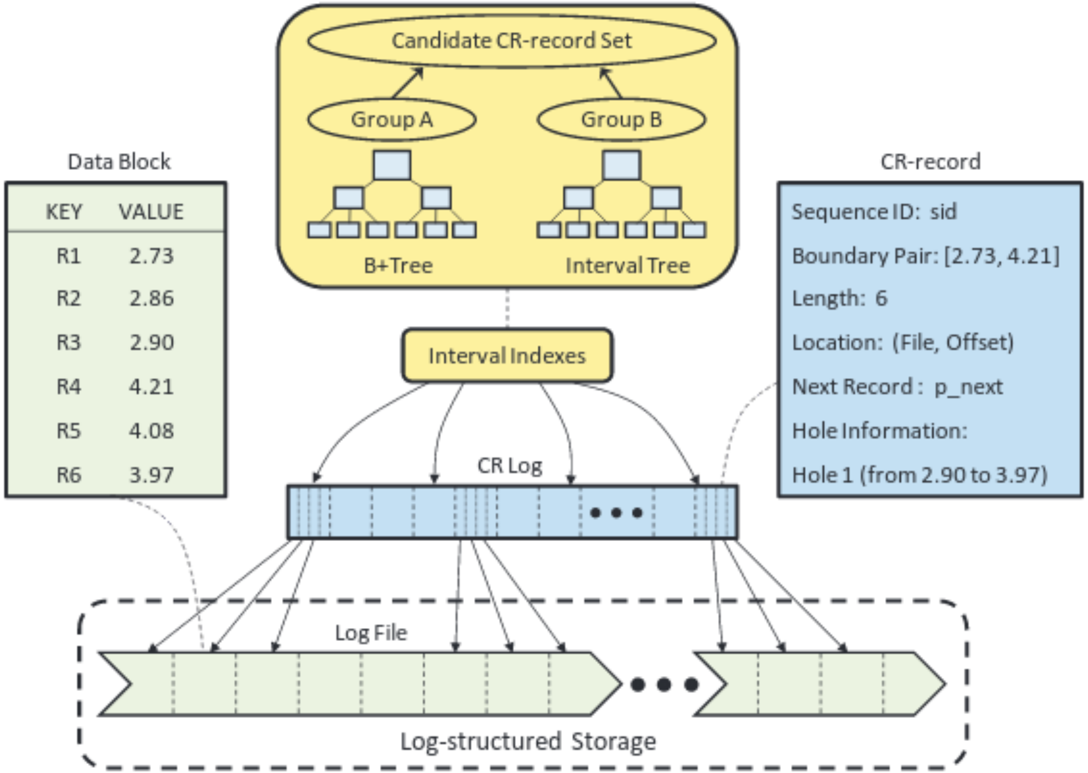
Images taken from the paper



KEY	ATTRIBUTE	VALUE	TIMESTAMP
depth 0m	Salinity	16.32	9:01
depth 0m	Oxygen	3.36	9:01
depth 2.4m	Salinity	22.38	9:01
depth 2.4m	Oxygen	3.28	9:01
depth 8m	Oxygen	2.97	9:02
depth 8m	Salinity	29.01	9:02
depth 0m	Salinity	16.14	9:02

Figure 3: Schema physical view

CR-Index



Data Block

KEY	VALUE
R1	2.73
R2	2.86
R3	2.90
R4	4.21
R5	4.08
R6	3.97

CR-record

Sequence ID: sid
Boundary Pair: [2.73, 4.21]
Length: 6
Location: (File, Offset)
Next Record : p_next
Hole Information:
Hole 1 (from 2.90 to 3.97)

Figure 4: The CR-index structure

* Image taken from the paper

CR-Index

- Low index maintenance is needed to keep high write performance.
- CR-Index uses locality traits of observational data to create a pruning based lightweight index.
    - Nearby records are bunched into blocks and described by a **boundary pair** [min, max].
    - A block spans multiple records and a log of blocks is maintained.
    - A record in CR-index is called a CR-Record and contain: ***block ID, boundary pair, hole information, block length and file position.***

Insertion

- Directly append the record to the log store.
- The authors don’t mention any optimum length of the number of records in a CR-Record or any optimized construction algorithm.

Queries

- **Point Queries are not made**.
- **Range Queries are transformed Intersection Checking**.
- If CR-Log fits into memory then a linear scan is performed.
- If CR-Log does not fit into memory, we need to optimize as interval based intersection queries tend to visit a lot of internal nodes.
- Each Range Query is divided into two sub-queries:

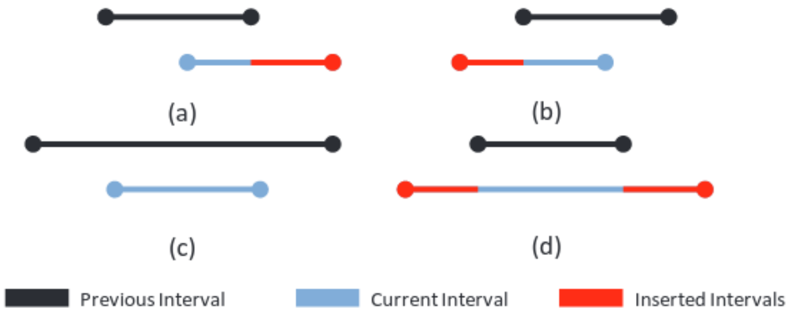
  Group A: CR-Records having at least one endpoint in query range.

  Group B: CR-Records containing the query range.

Range Queries

- **Group A**: A B+-tree is used. Both the end-points of each CR-Record is inserted into the tree and for a range query **[a,b]** we find one endpoint and do a scan.
- **Group B:** A stabbing query representing the intersection query is used. Maintain a segment tree of intervals and search a point **d** in between **[a,b].**
- Join results from **A** and **B**.
- So, for every query only one path is taken in the tree.

Index Optimizations

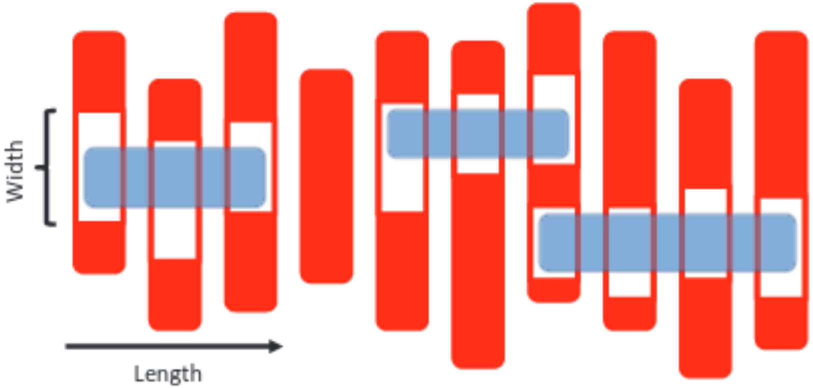


Delta Intervals

- Need to insert 2 * CR-Records to B+-tree.
- This can be reduced by only inserting the interval difference between current block and previous block and scanning the next block for each result.
- can be extended to previous k-blocks

* Image taken from the paper

Index Optimizations



Hole Skipper

- The continuity assumption may not be valid
- Each CR-Record maintains information about only top **k** holes (to make records smaller).
- Blocks with false positives have holes and these holes are populated at query time to not degrade write performance.

* Image taken from the paper

Disordered Inserts

- Network delays may lead to disorder inserts.
- The system inherently doesn’t care about order within a CR-Record and disorder across blocks is handled by the hole skipper.
- For time based queries, a memory based checkpoint list is created which is used to find time ranges to search in.

Attribution

- All figures used in this presentation have been taken from http://www.vldb.org/pvldb/vol7/p529-wang.pdf.

Questions