Indexing for Interactive Exploration of Big Data Series

Kostas Zoumpatianos, Stratos Idreos, Themis Palpanas

SIGMOD'14

曾丹 2014.10.23

Outline

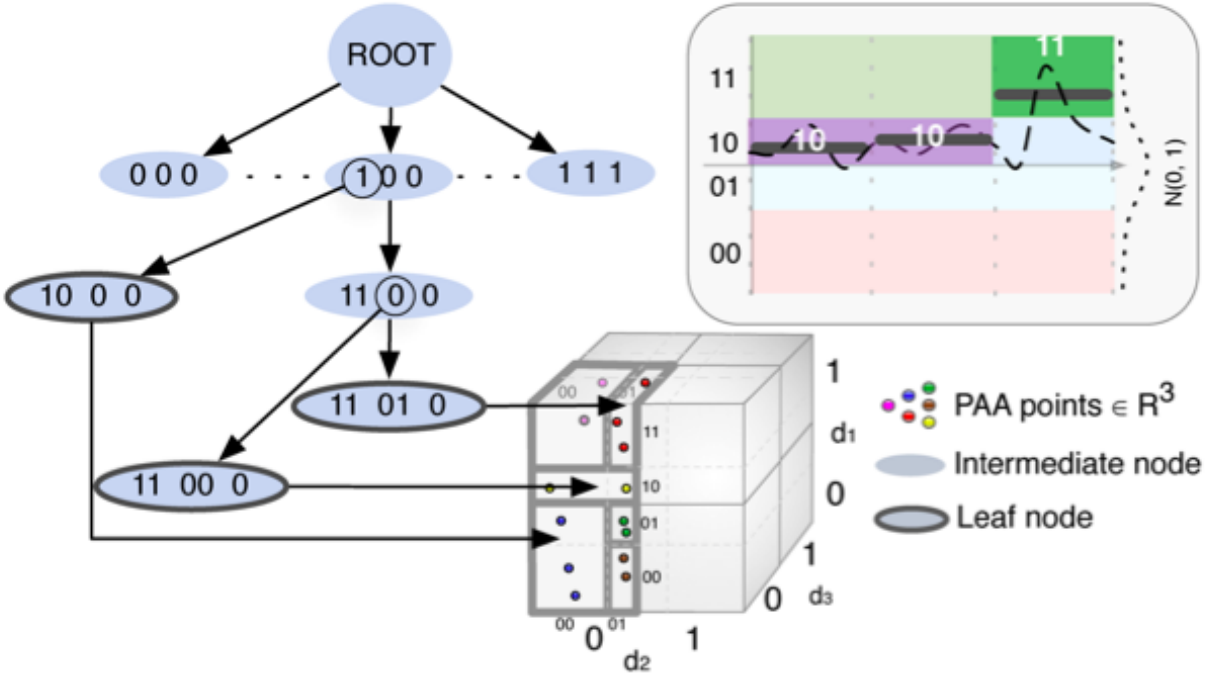
- Background
- ADS/ADS+/PADS+
- Evaluation
- Related Work
- Conclusion

Background

- Data series
  - $T = (p_1, \ldots, p_n)$ $p_i = (v_i, t_i)$
  - Web usage data, weather data, stock data, etc
  - Examine the sequence of values instead of single points
- Exploratory similarity search in data series
  - Data exploration
    - Need to build index to efficiently process query
    - the cost of building an index is a significant bottleneck
  - Similarity search
    - One of the most basic data mining tasks
    - Dimensionality reduction
- Adaptive indexing
  - Build index during query processing
  - More than one column in case of similarity search

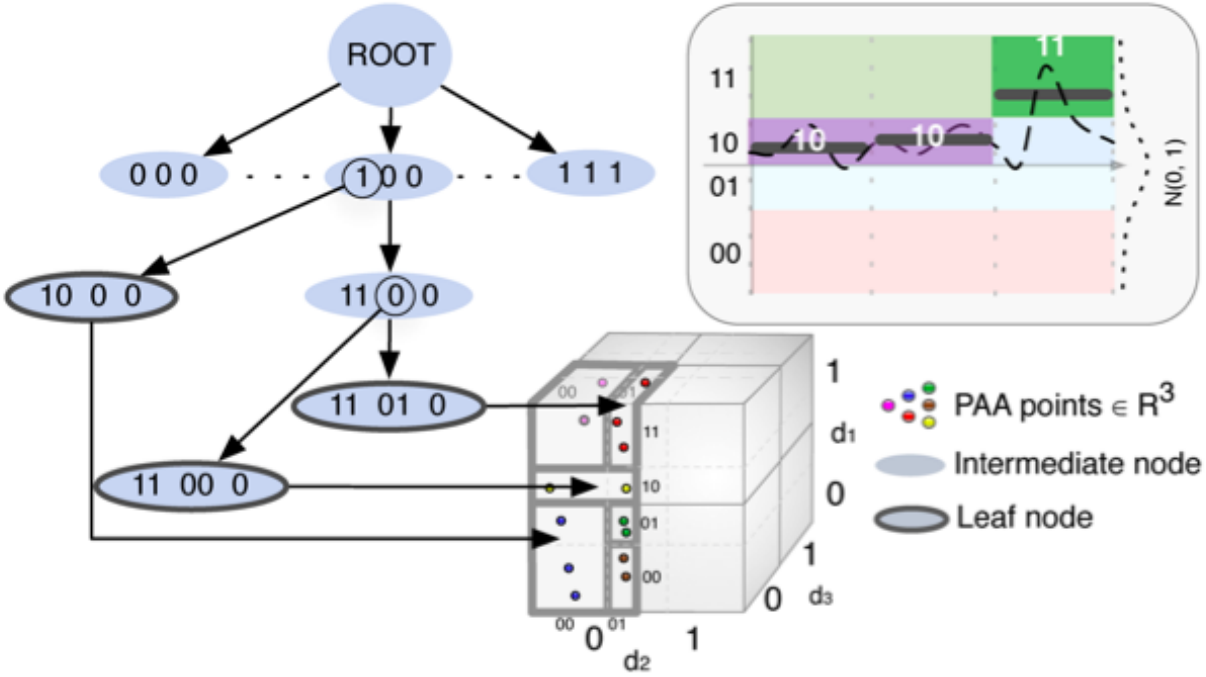
Background

- Dimensionality reduction
  - PAA(time dimension)
  - SAX(value dimension)



Background

- iSAX
  - $(character)_{cardinality}(character)_{cardinality}(character)_{cardinality}$
  - $00_2 10_2 01_2 , 00_2 11_2 01_2 \Rightarrow 00_2 1_1 01_2$ (reduction on the second character)



The Adaptive Data Series Index

- The ADS Index
- The ADS+ Index
- Partial ADS+ Index

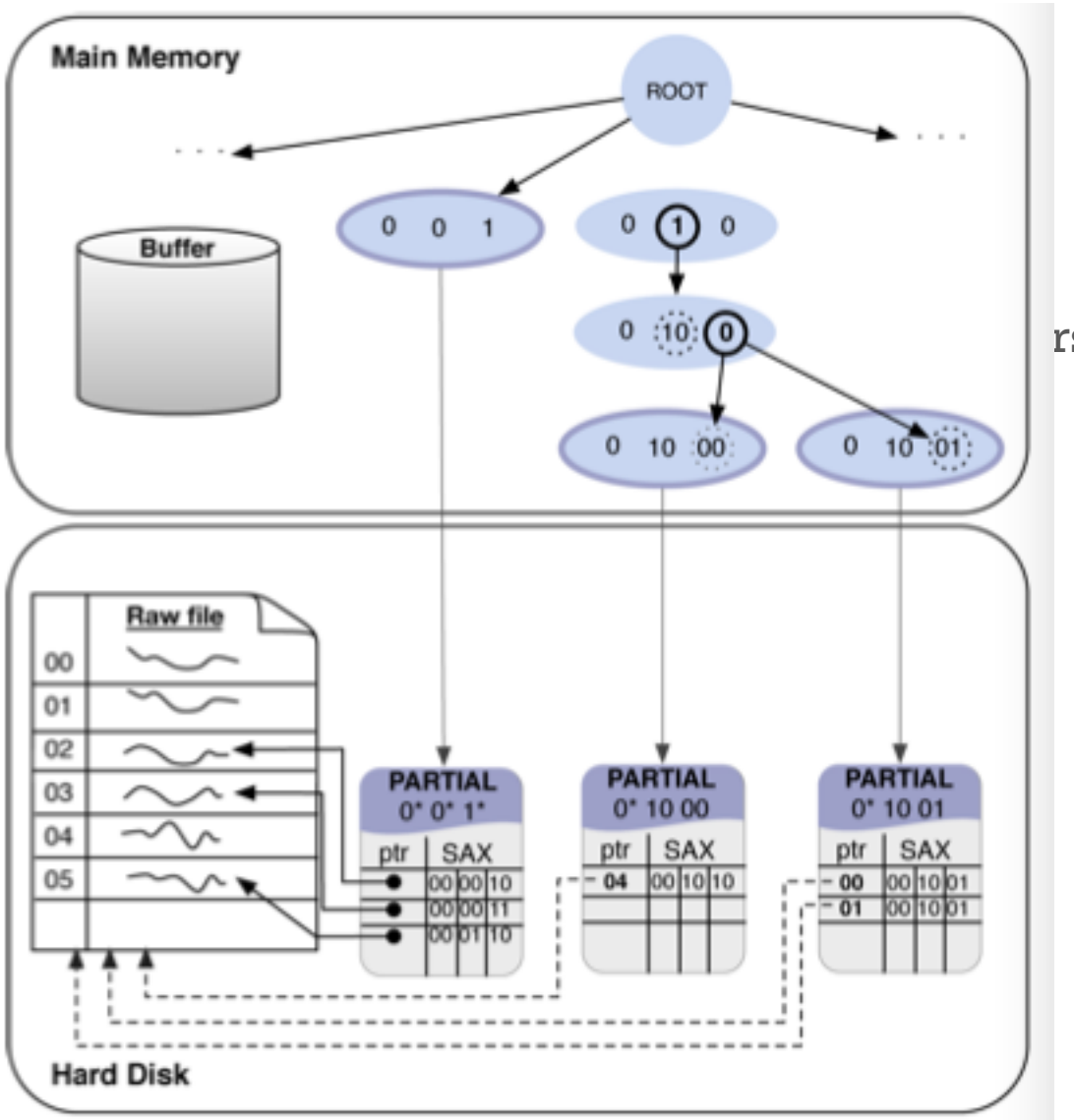
ADS

- Motivation
  - iSAX 2.0 index building cost
    - Read raw data series from disk and write the leaves of the index tree
    - Build index, then query data
- ADS
  - Index creation phase
    - Create a tree that contains only the iSAX representation for each data series
  - Query time
    - Only load relevant data from raw data files

ADS

- Index creatio
  - Read raw da
    in FBL buffer
  - When memo
    - If the targe
  - Flush LBL bu
  - Set leaf in PA

rs



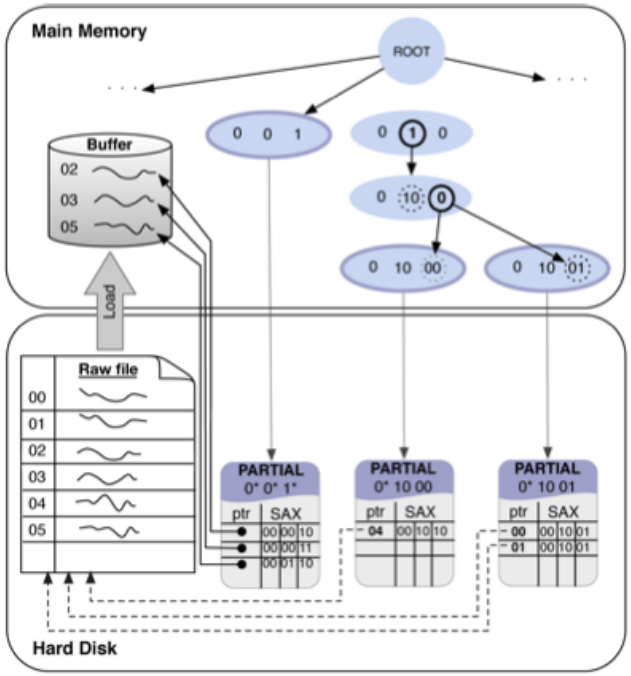
(a) Initial state

ADS

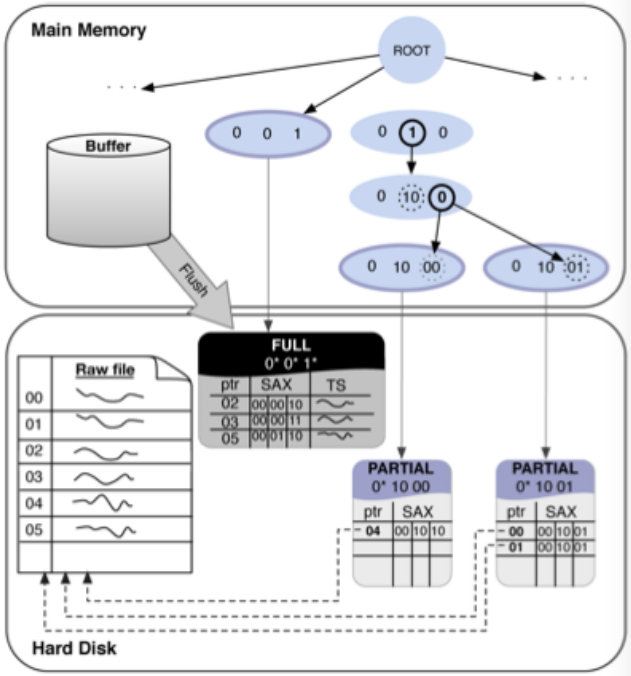
- Delaying Leaf Construction
  - Reduce split cost by avoid moving raw data series through the tree
  - Reduce write cost of raw data files during index phase
- Buffering
  - Write to disk one leaf at a time => sequential writes??
- Mapping on raw data files
  - Maintains positions to get raw data series in query time

ADS

- Querying and refining ADS
  - Search index
  - Enrich index
  - Create answer



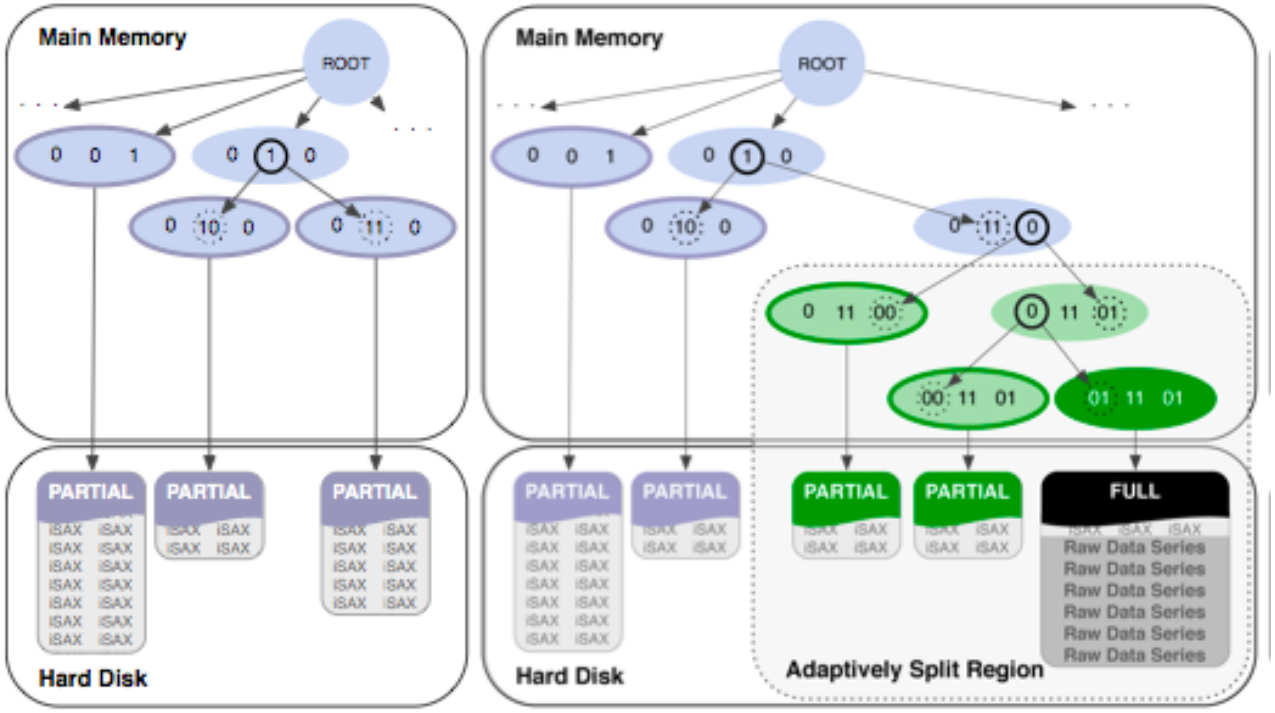
(b) Buffered leaf state



(c) Flushed leaf state

ADS+

- Motivation
  - time spent during split operations in the index tree is a major cost component
- Leaf size
  - Big leaf size
    - Reduce tim
  - Small leaf siz
    - Read less
  - Adaptive
    - a big build
    - A small qu



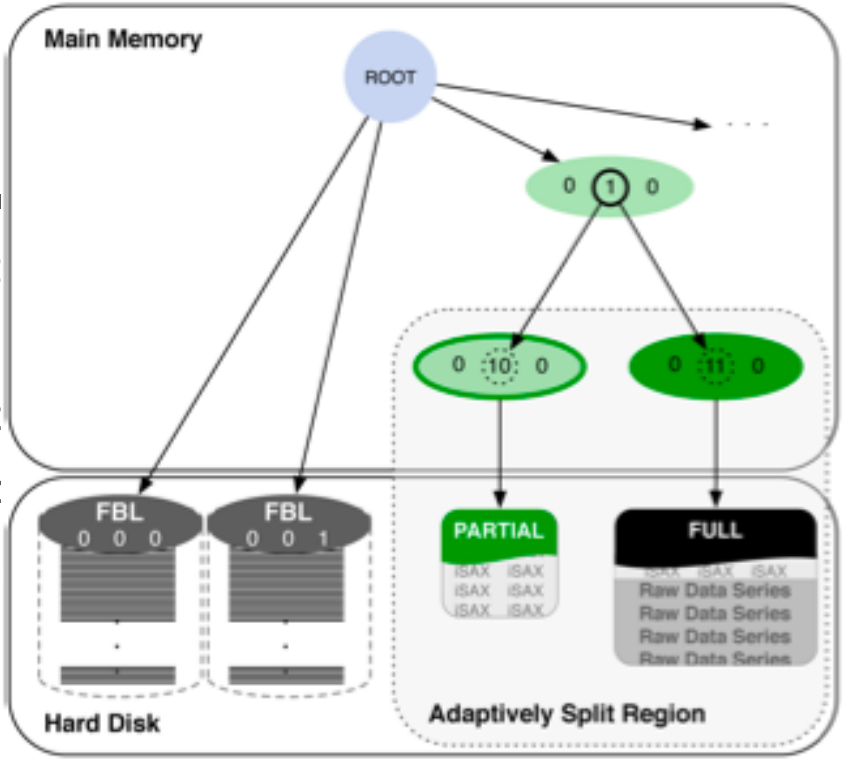
(a) ADS+ after index building. (b) ADS+ index after a query.

ADS+

- Only create fine-grained version of the sub-tree related to current workload
- Less split operations => less computation cost
- Smaller iSAX representations of the unrelated data => less I/O
- Only materialize related leaf nodes => better adaptive behavior

PADS+

- Motivation
  - ADS and ADS+ still has to wait for creating the basic index tree
- Methodology
  - Initialization phase
    - Create a root node and a set of FBL
    - When FBL buffer is full, flush it to d
  - Query time
    - Read corresponding FBL buffer fro
    - Continuously split it until query-tim
    - Load raw data files from correspon approximate answer



Updates

- Inserts
  - appending the new data series in the raw file
  - Only (iSAX representation, position) pair is pushed through the index tree
  - If the leaf is in full mode, flip a bit in this leaf so that future queries know that more data exists.
  - fetches the new inserts on-the-fly and merges them
- Deletes
  - Mark the data series as deleted
  - In query time, ignore the deleted data series

Evaluation

- Algorithms
  - ADS, ADS+, PADS+, iSAX 2.0, buffered iSAX 2.0, R-Trees, X-Trees
- Infrastructure
  - C, GCC 4.6.3, linux 12.04.2
  - An Intel Xeon machine(64GB RAM; 4x 2TB, SATA, 7.2K RPM Hard Drives in RAID0)
- Benchmarks
  - Data to search
    - Synthetic benchmarks(N(0,1)) and real-life benchmarks
    - Data series: 256 points with 4 bytes value each
  - Query
    - Query intensive workloads as well as updates
    - Various workload patterns including skewed workloads

Reducing the Data to Query Time

500 million data series
10^5 random queries
(73% would fetch new raw data)

Index building cost

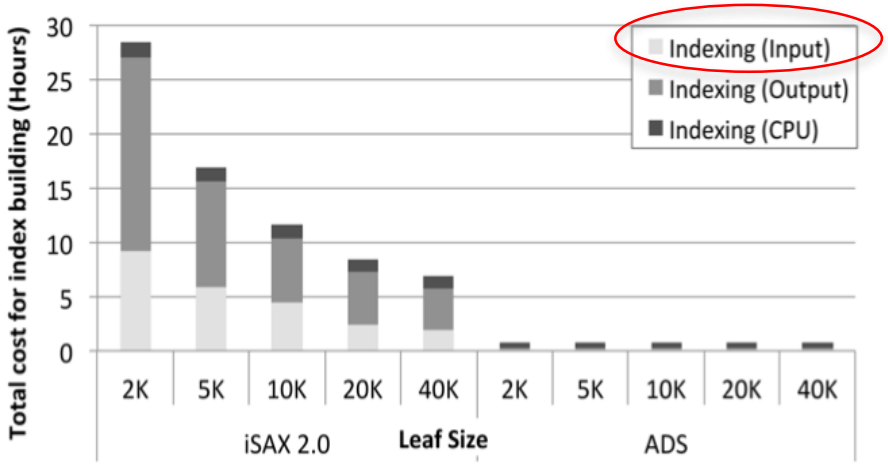


Figure 6: Reducing indexing costs.

I/O and cpu cost have significantly decreased

Query processing bottleneck of ADS

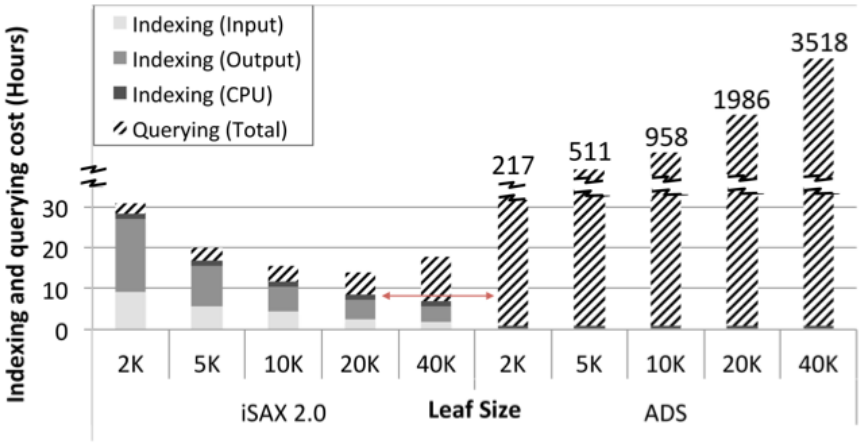


Figure 7: The query processing bottleneck.

Random workloads might result in significant amount of raw data series

Reducing the Data to Query Time

500 million data series
10^5 random queries
(73% would fetch new raw data)

Robustness with ADS+

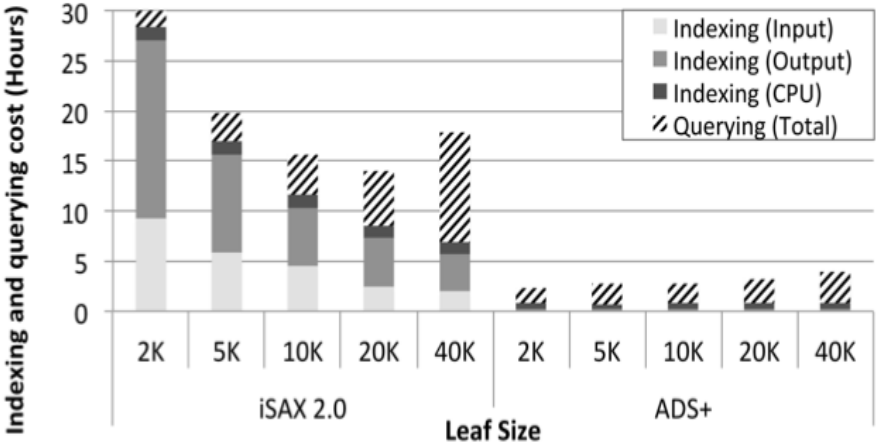


Figure 8: Reducing the data-to-query time with ADS+.

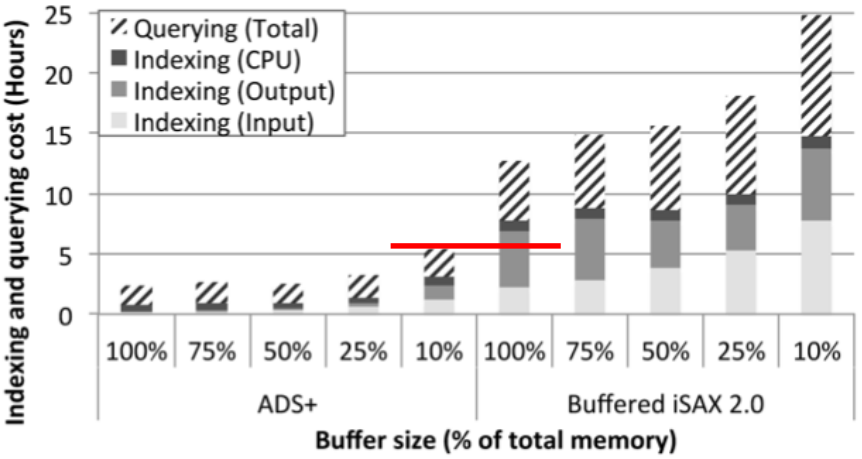


Figure 9: Total indexing and query answering cost as we increase the buffer size for ADS+ and iSAX 2.0.

ADS+ outperfoms iSAX 2.0 during index building phase and querying processing phase

ADS+ can answer all the queries before iSAX 2.0 has finished indexing

Reducing the Data to Query Time

500 million data series
10^5 random queries
(73% would fetch new raw data)

- Choosing the Query-Time Leaf Size

Query-time leaf size	1	10	100	1000
Query time (millisecond)	11.27	67.64	499.95	4031.68
Number of 4KB pages	0.25	1.79	17.23	171.48
Number of 8KB pages	0.12	**0.89**	8.61	85.74
Number of 16KB pages	0.06	0.45	4.30	42.87

Table 1: Varying query-time leaf size.

Only considering time?

Smaller query-time leaf size => less data to fetch, faster materialization of the leaf node
Smaller query-time leaf size => smaller page utilization

Reducing the Data to Query Time

10^5 random queries
(73% would fetch new raw data)

- Scalability

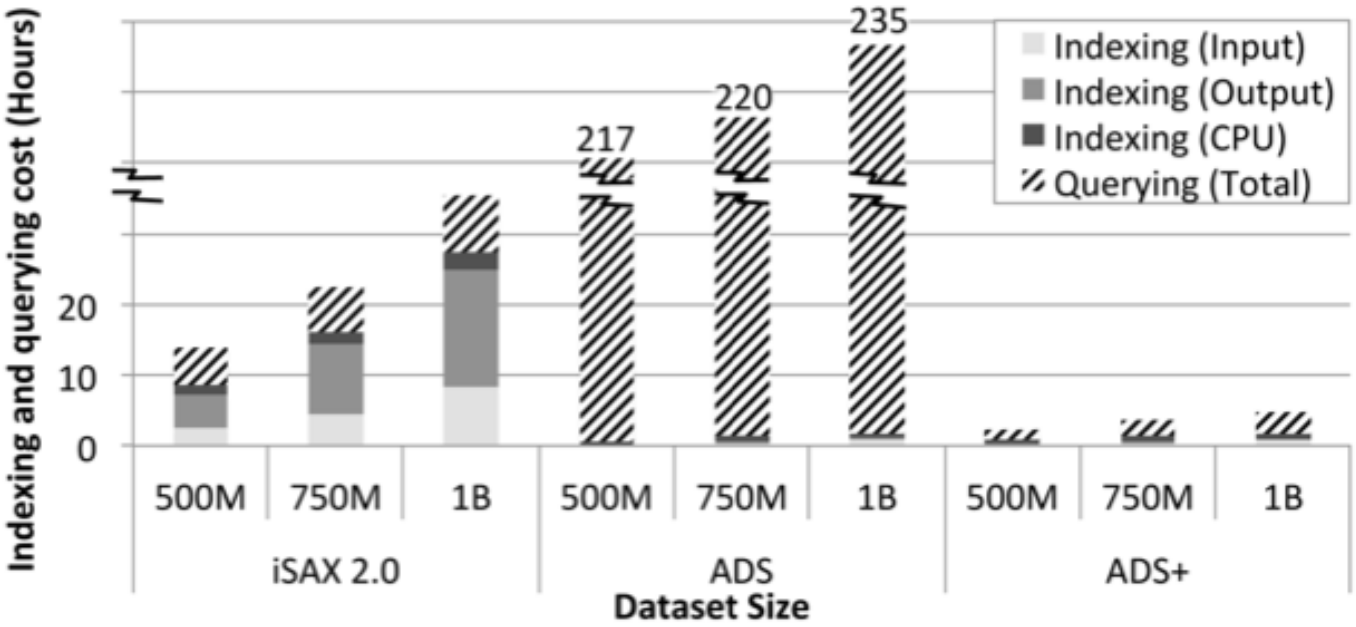
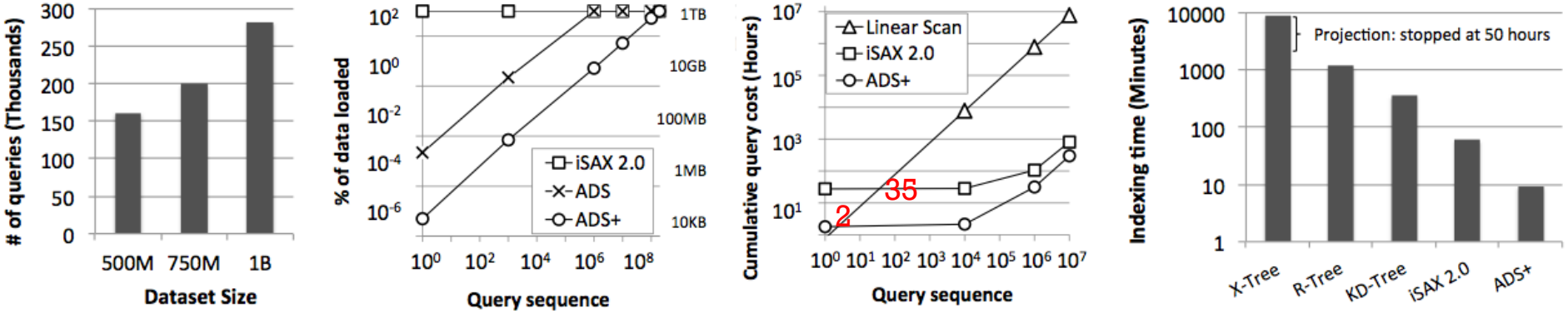


Figure 10: Scaling to 1 billion data series.

ADS+ significantly outperforms all other strategies

Reducing the Data to Query Time

- Scalability



(a) Queries answered by ADS+ while iSAX 2.0 is still indexing. (b) Percentage of data indexed. (c) Per query response time. (d) ADS+ against spatial indexes.

Figure 11: Reducing the data-to-query time with ADS+ as we scale to big data.

I/O and cpu cost have significantly decreased

Adaptive behavior under updates

100 million data series
10^5 random queries
(73% would fetch new raw data)

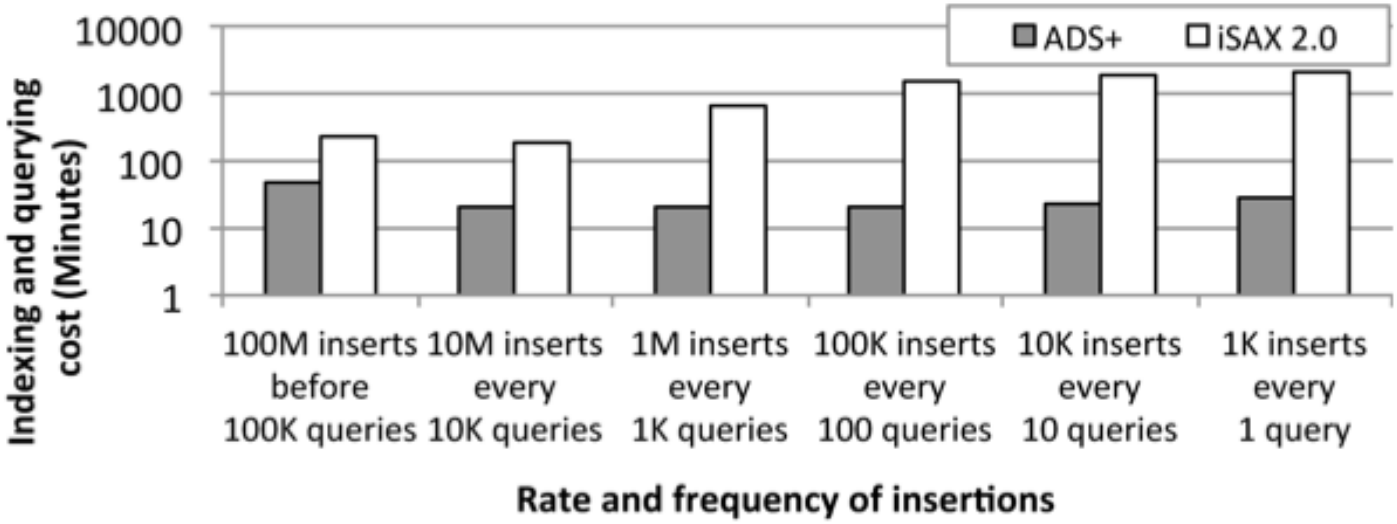
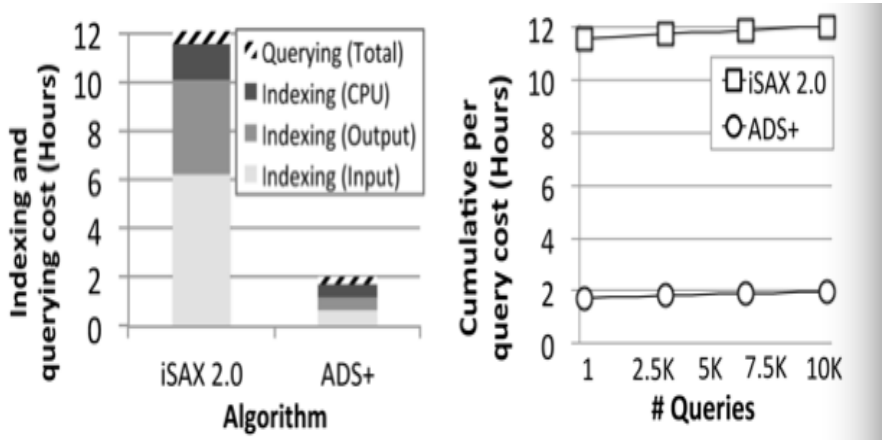


Figure 12: Updates for 100 million data series and 100 thousand queries in 6 different batch sizes.

ADS+ has better adaptive behavior and better performance

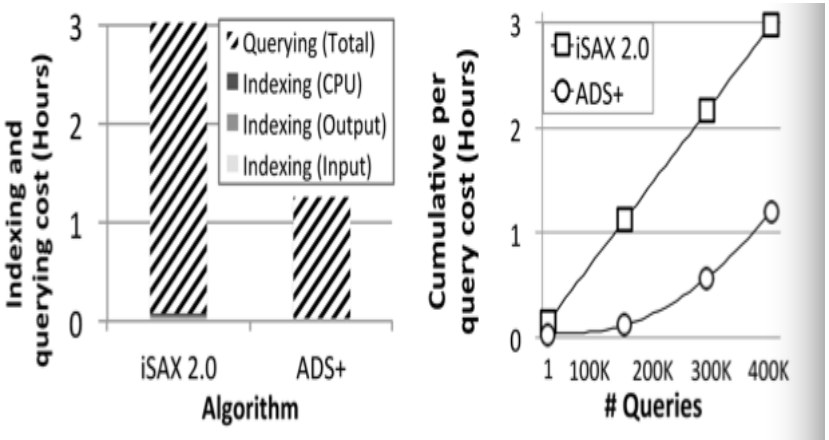
Real-life Workloads



(a) Total costs.

(b) Cumulative per query costs (Query 1 includes indexing).

Figure 13: Indexing 1 Billion images (SIFT vectors) and answering 10^4 queries.



(a) Total costs.

(b) Cumulative Per query costs (Query 1 includes indexing).

Figure 14: Indexing 20 Million DNA subsequences from the Homo Sapiens genome and answering $4 * 10^5$ queries.

ADS+ outperforms iSAX 2.0 in indexing and querying

PADS+

1 billion data series
10^4 queries

Workload	Cross-over point (PADS+ over ADS+)
Random	2899 queries
Low skew	2970 queries
Medium skew	3097 queries
High skew	3825 queries

Table 2: Fast access with PADS+ with varying skew.

Low skew: 60% queries are picked from 40% of the domain
Medium skew: 80% queries are picked from 20% of the domain
High skew: 99.99% queries are picked from 0.01% of the domain

PADS+ is the best choice in case of skew workload

Related Work

- Similarity Search
  - Dimensionality reduction
    - DFT, DWT, DHWT, PAA, SAX
  - Distance measures
    - DTW, ED
- Adaptive indexing
  - Column-store databases
    - Focus on how to incrementally sort columns
    - The query predicates are used as pivots during index refinement
    - Range index instead of tree-structure based index
    - Index only one column
- Scan vs indexing
  - [1] have shown sequential scan can be performed efficiently
    - Applied to the database with a single, long data series and small subsequences match
    - Indexing is required to support data exploration tasks

[1] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and miningtrillions of time series subse- quences under dynamic time warping. In *SIGKDD*, pages 262–270, 2012.

Conclusion

- An adaptive indexing method on data series
  - Avoid storing raw data in leaves
  - Adaptive leaf size
  - Only indexing relevant data