Access Path Selection in Main-Memory Optimized Data Systems: Should I Scan or Should I Probe?

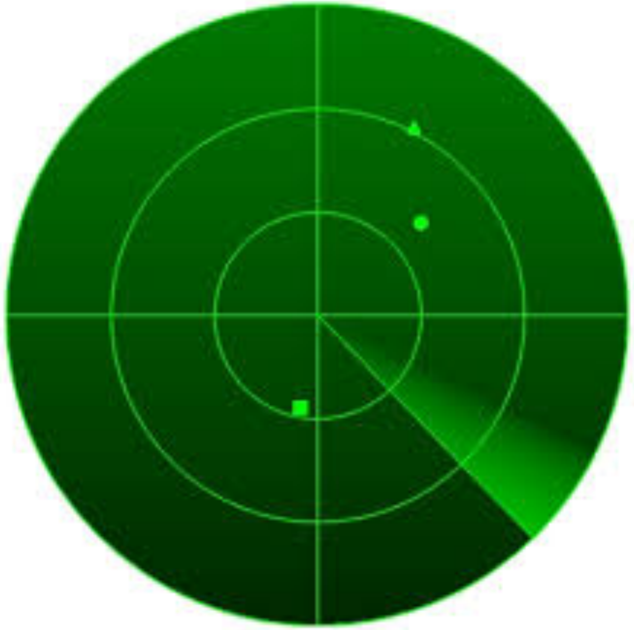
CS 591A: Data Systems Architecture

February 26th, 2019

Aleksandr Kim

Content

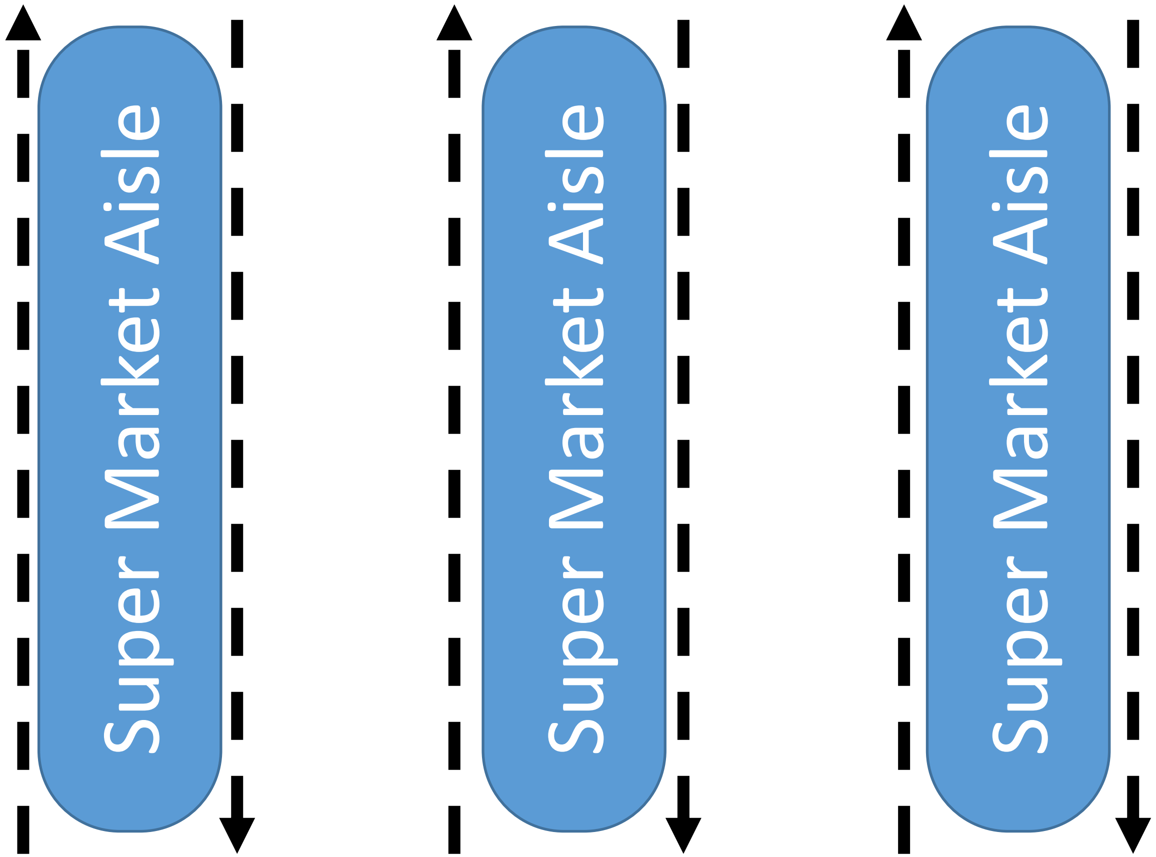
- Background
- Motivation and Summary
- Paper Methodology
- Analytical Model Details
- Analytical Model Results
- Experimental Model
- Final thoughts



Access Path Selection: Scanning and Probing?

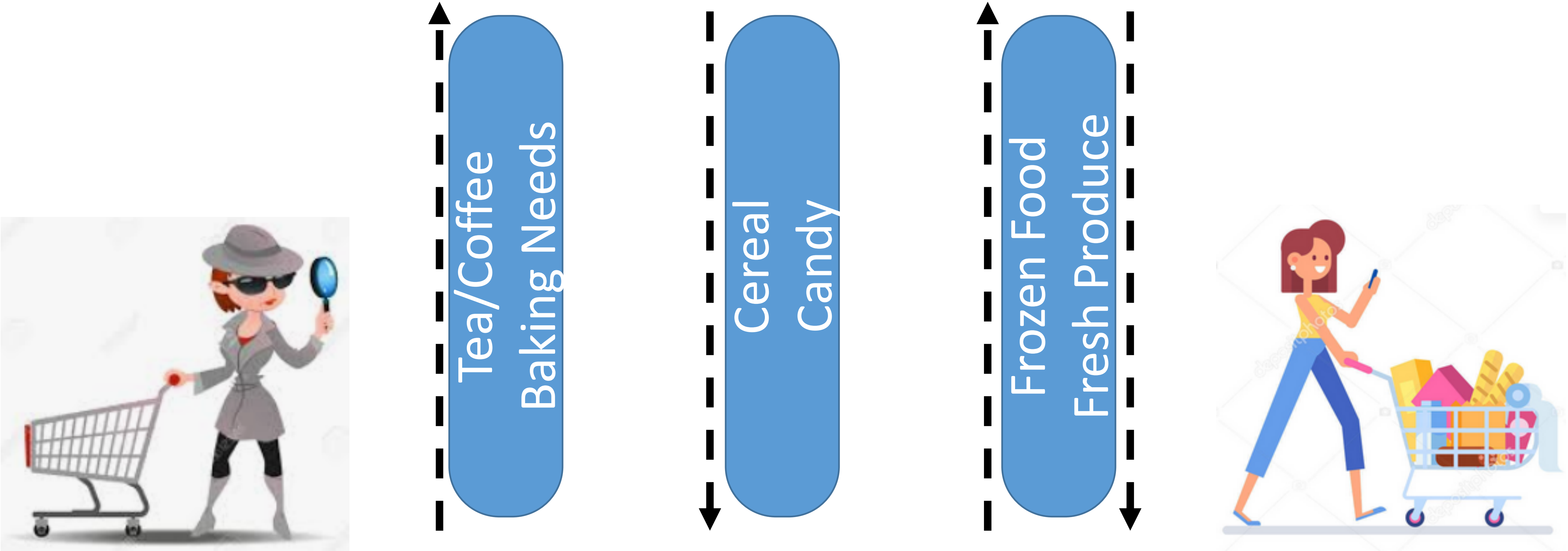
Access Path Selection: Scanning and Probing?

Scanning: Sequential search through all the data



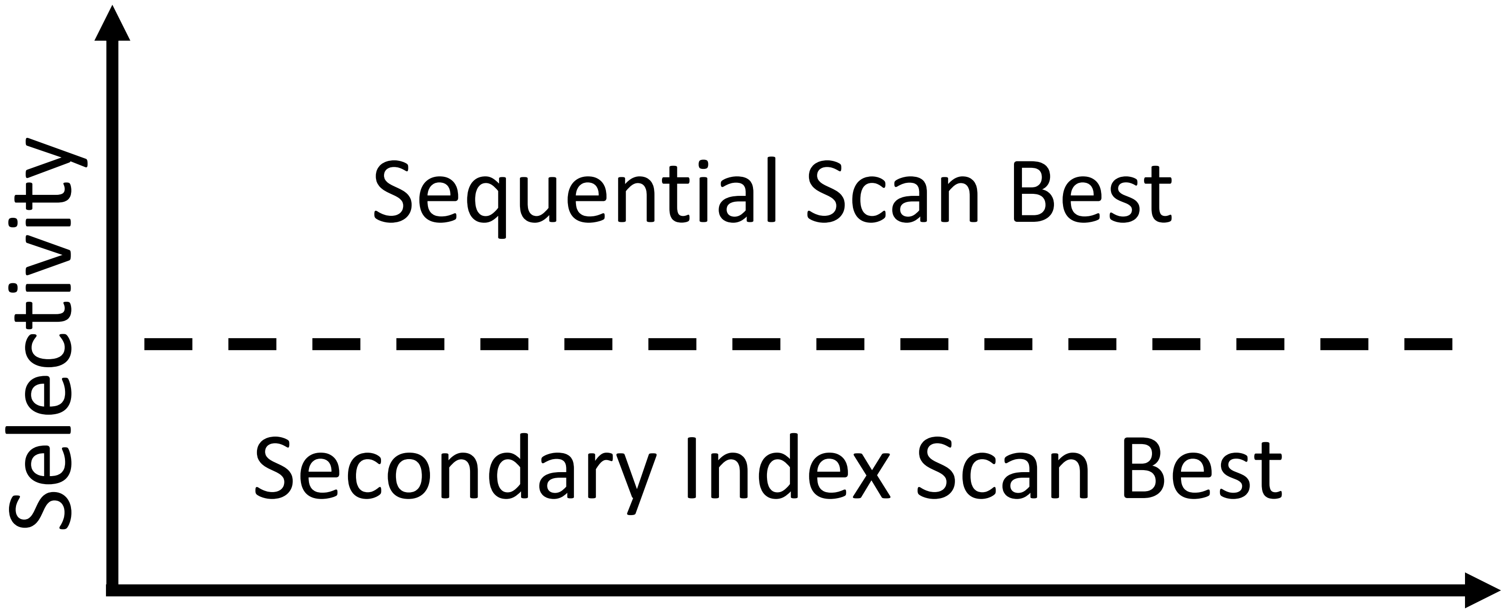
Access Path Selection: Scanning and Probing?

Probing, Secondary Index Scan: Sort data and search



Historic Approach to Access Path Selection

- Selection of best Access Path based on a variety of factors but largely dependent on query selectivity
  - The more stuff you have to get, the more stuff you have to look through



Access Path Selection? Performance is Key

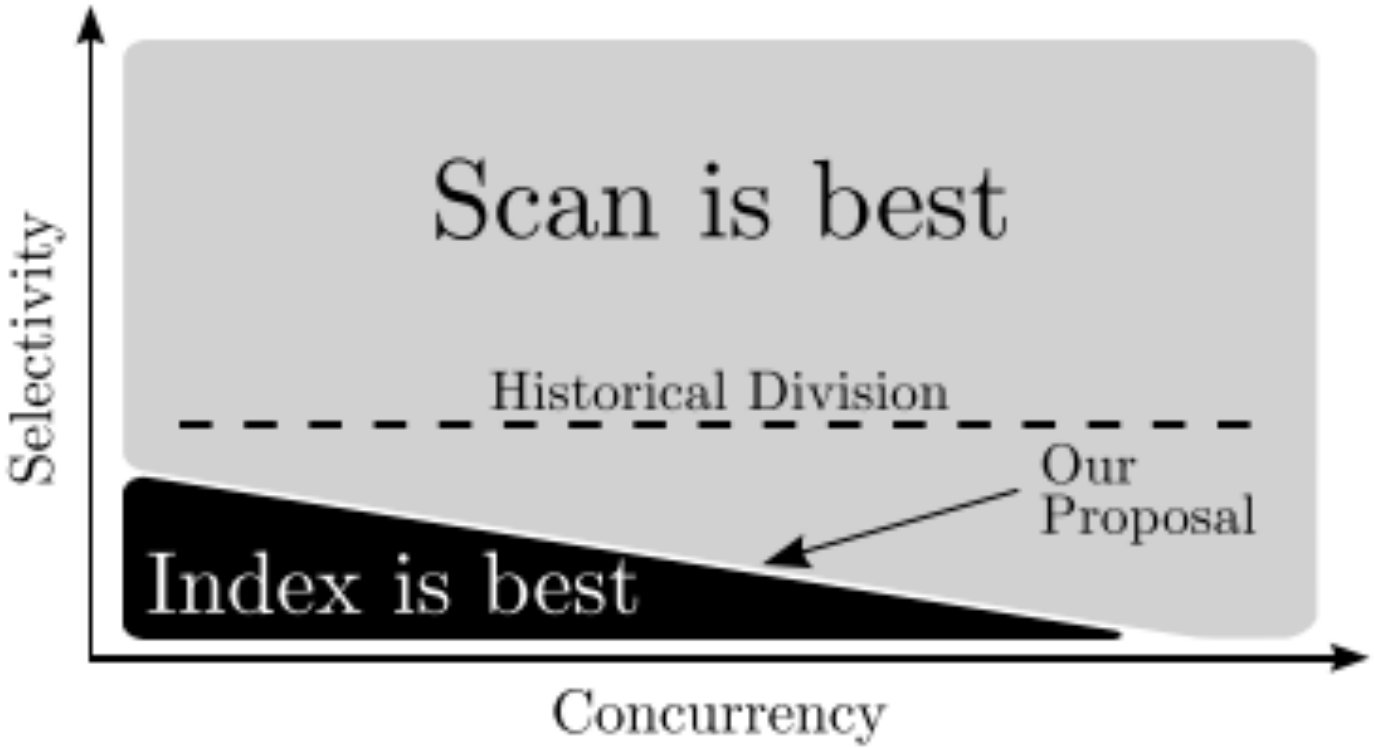
Scan operator greatly optimized over the years

- Rise of column data storage
- Improved hardware capability
- Parallel computing
- Data compression

- Possible to hide minor inefficiencies
- Selection of access path might not even matter

Summary: Access Path Selection Still Important!

- Selection based on query concurrency in addition to selectivity
- Possible to create a system that dynamically switches between sequential and secondary index scanning



Methodology: Analytical and Experimental

- Mathematical model created to analyze scan performance
  - Workload
  - Data properties
  - Hardware specifications
  - Data structure characteristics
- Prototype data system, FastColumns, built and tested
  - Four hardware configurations

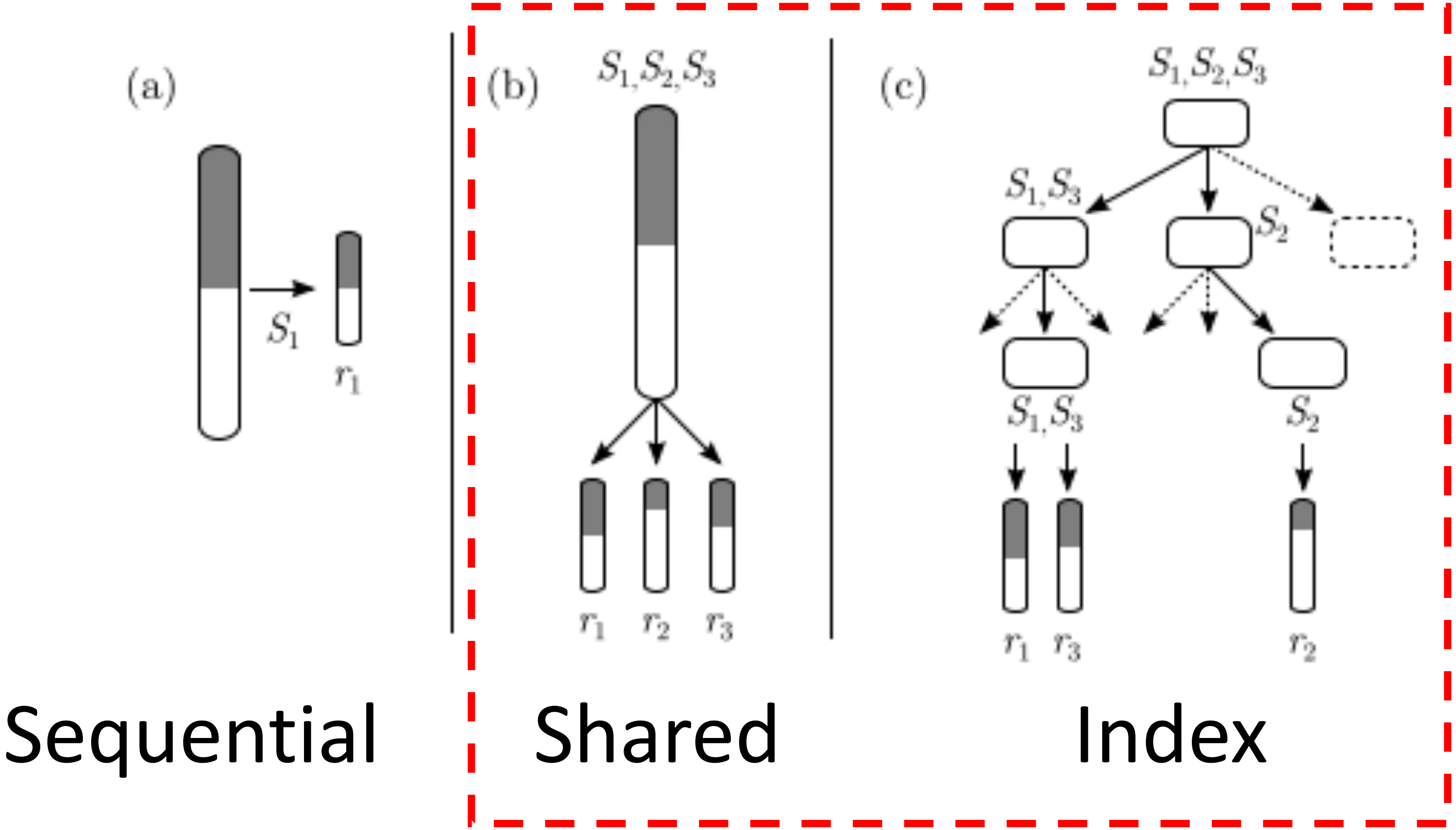
Analytical Model

- Comprehensive mathematical model on the select operator
  - Inputs point or range query
  - Outputs collection of rows
- Other enhancements
  - Data compression: Dictionary compression to 2 bytes
  - Zonemaps: Group data to reduce tuples searched

Workload	q	number of queries
	s_i	selectivity of query i
	S_{tot}	total selectivity of the workload
Dataset	N	data size (tuples per column)
	ts	tuple size (bytes per tuple)
Hardware	C_A	L1 cache access (sec)
	C_M	LLC miss: memory access (sec)
	BW_S	scanning bandwidth (GB/s)
	BW_R	result writing bandwidth (GB/s)
	BW_I	leaf traversal bandwidth (GB/s)
	p	The inverse of CPU frequency
	f_p	Factor accounting for pipelining
Scan & Index	rw	result width (bytes per output tuple)
	b	tree fanout
	aw	attribute width (bytes of the indexed column)
	ow	offset width (bytes of the index column offset)

Analytical Model

- Focus on shared and index scans



Access Path Selection

$$APS = \frac{ConcIndex}{SharedScan}$$

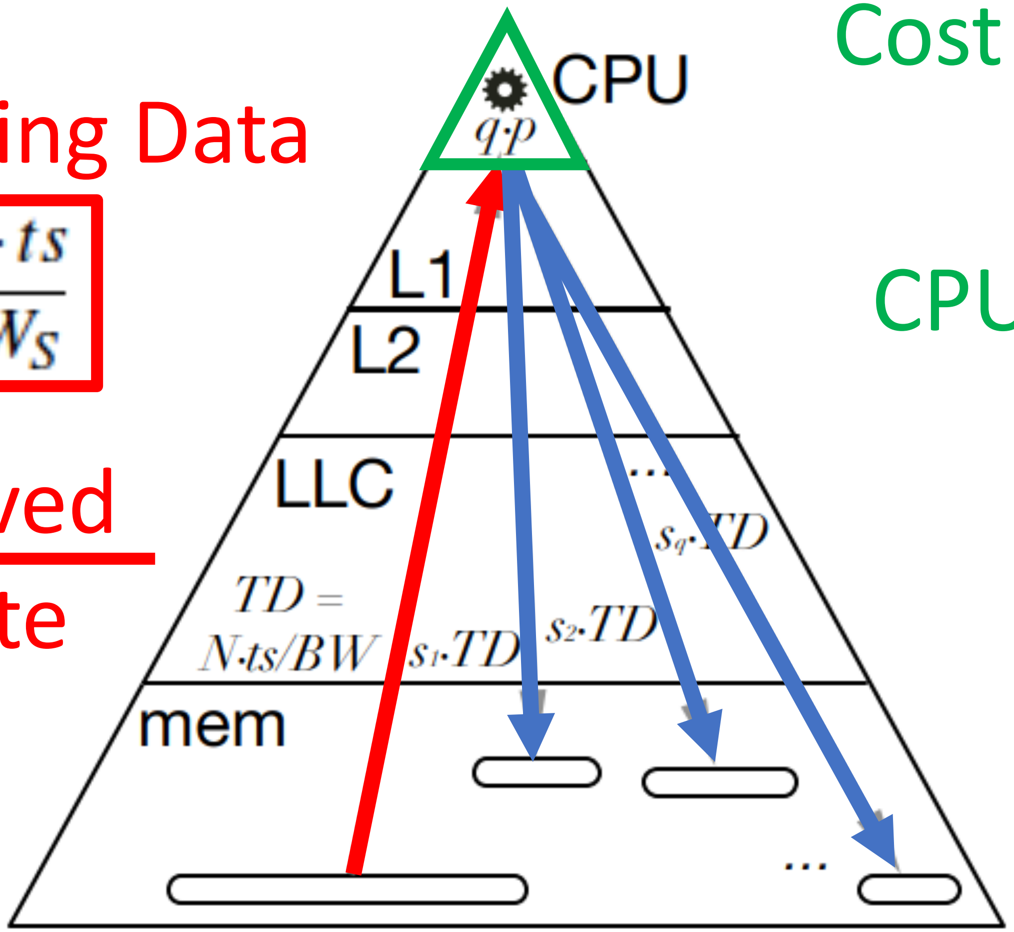
Can decide based on what will be less costly

Shared Scan Model

Cost of Moving Data

$$TD_S = \frac{N \cdot ts}{BW_S}$$

$$\frac{\text{Data Moved}}{\text{Scan Rate}}$$



Cost of Predicate Evaluation

$$PE = 2 \cdot f_p \cdot p \cdot N$$

CPU Speed · "Calculations"

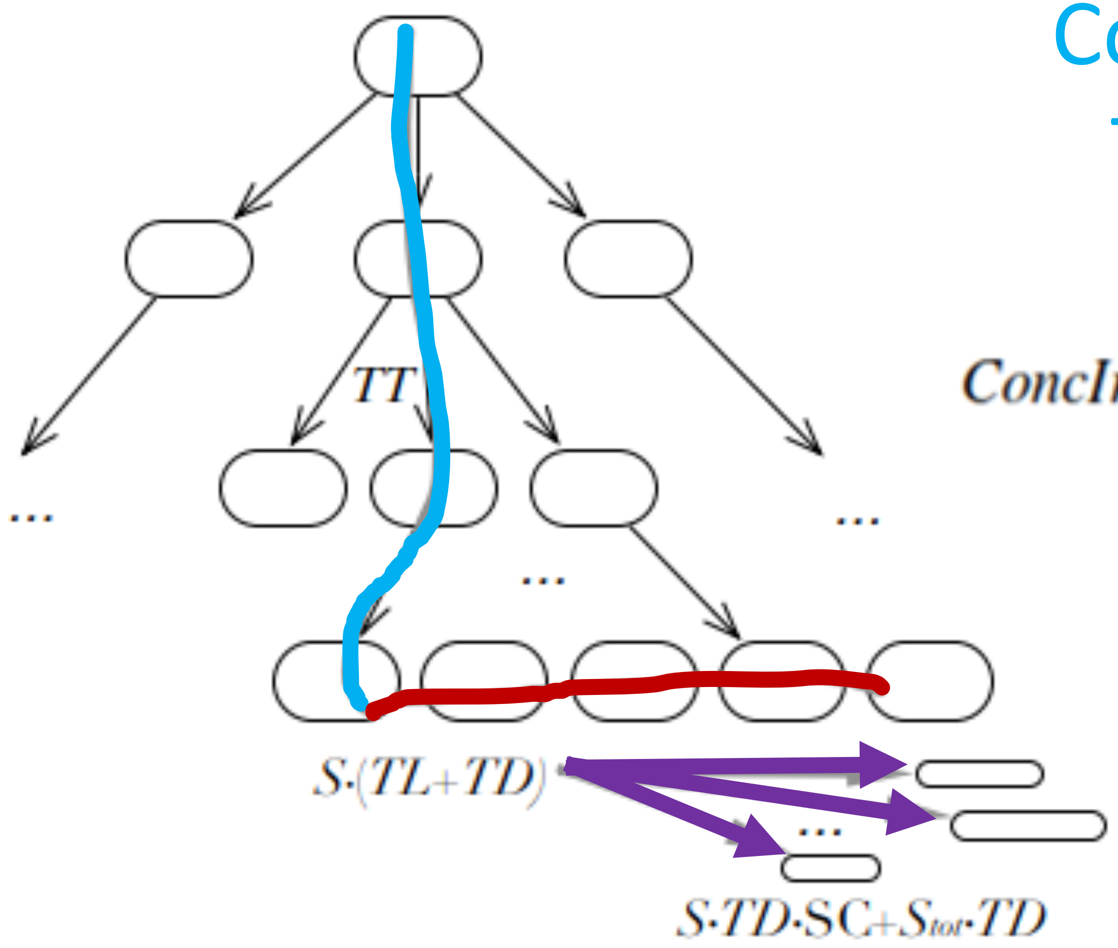
Cost of Outputting Data

$$TD_R = \frac{N \cdot rw}{BW_R}$$

$$\frac{\text{Data Found}}{\text{Write Rate}}$$

$$SharedScan = max(TD_S, q \cdot PE) + S_{tot} \cdot TD_R$$

Index Scan Model



Cost of Tree Traversal

Cost of Writing

$$ConcIndex = q \cdot TT + S_{tot} \cdot (TL + TD_I) + S_{tot} \cdot TD_R + SF \cdot C_A$$

Cost of Leaf and Data Traversal

Cost of Sorting

Analytical Model Predictions

- Expanding the Analytical Model: $APS = \frac{ConcIndex}{SharedScan}$

Function of Concurrency

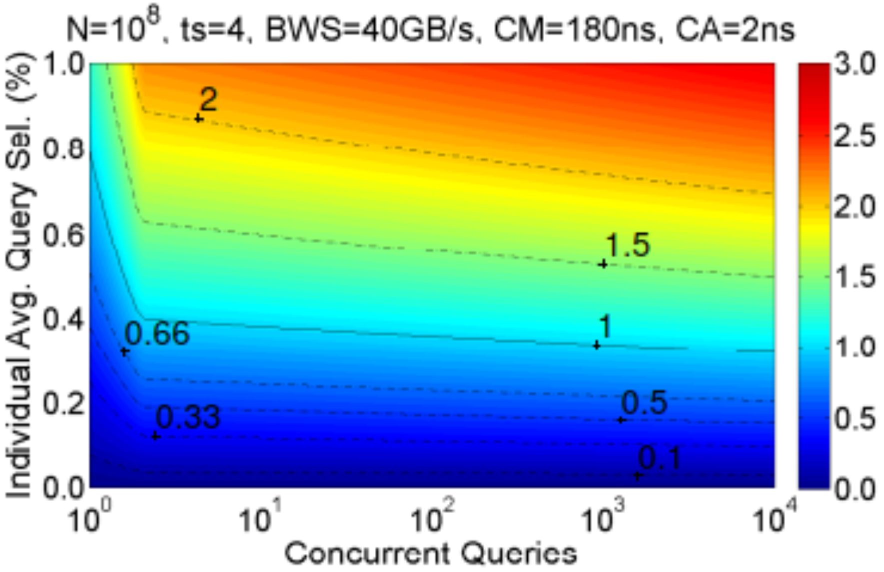
$$APS(q, S_{tot}) = \frac{q \cdot \frac{1+\lceil log_b(N) \rceil}{N} \cdot \left(BW_S \cdot C_M + \frac{b \cdot BW_S \cdot C_A}{2} + \frac{b \cdot BW_S \cdot f_p \cdot p}{2} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

$$+ \frac{S_{tot} \left(\frac{BW_S \cdot C_M}{b} + (aw + ow) \cdot \frac{BW_S}{BW_I} + rw \cdot \frac{BW_S}{BW_R} \right)}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}} + \frac{S_{tot} \cdot log_2\left(S_{tot} \cdot N\right) \cdot BW_S \cdot C_A}{max\left(ts, 2 \cdot f_p \cdot p \cdot q \cdot BW_S\right) + S_{tot} \cdot rw \cdot \frac{BW_S}{BW_R}}$$

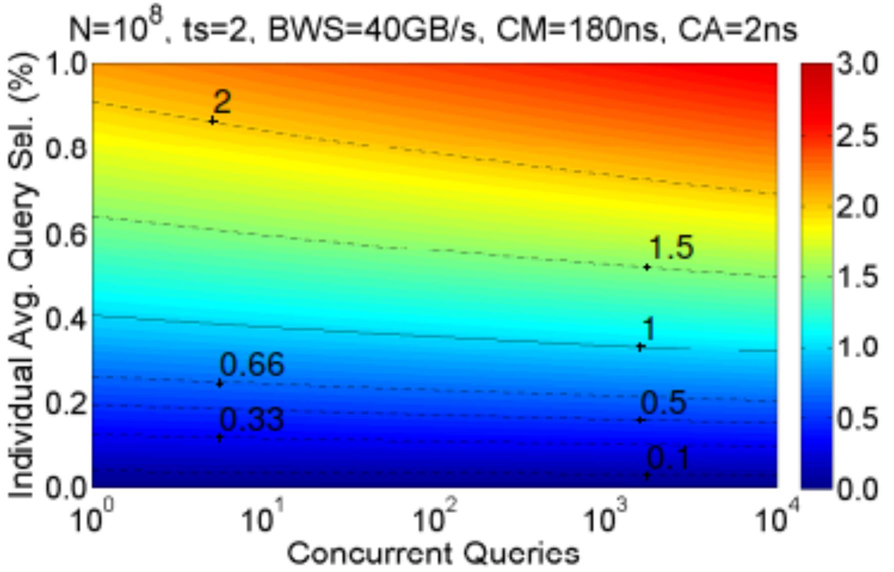
Function of Selectivity

Analytical Results Dependent on Concurrency

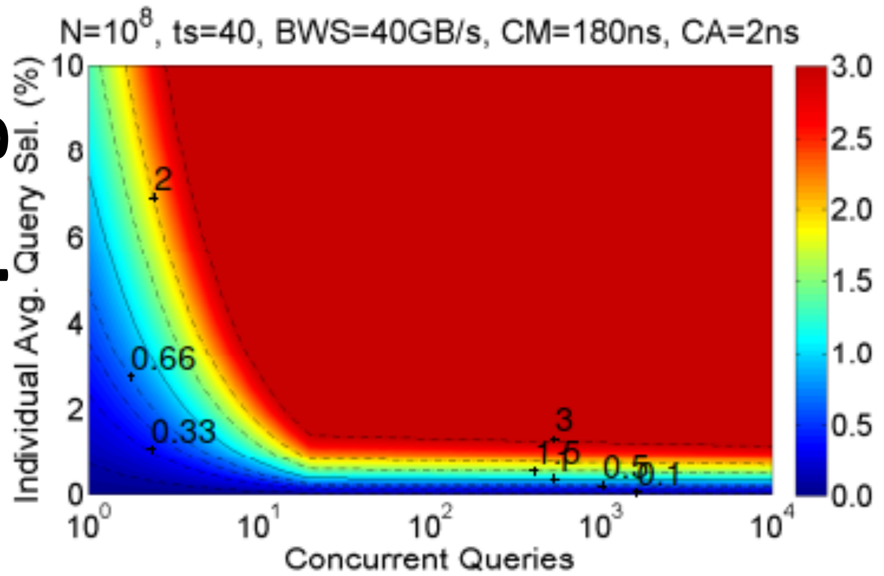
Baseline



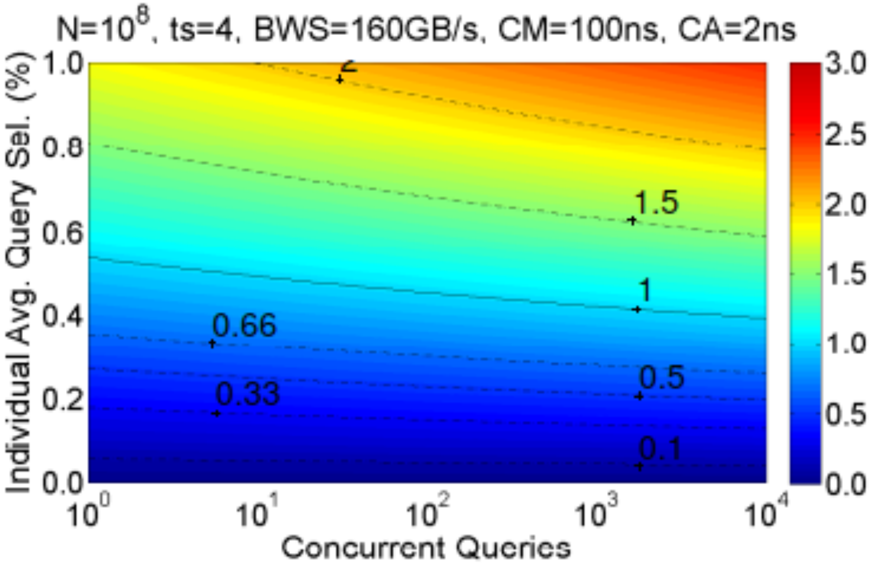
Compression



Column Grouping



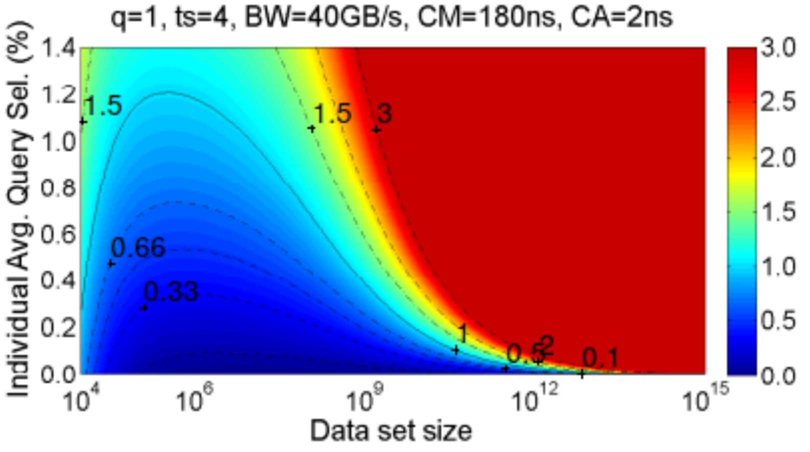
Alternate Hardware



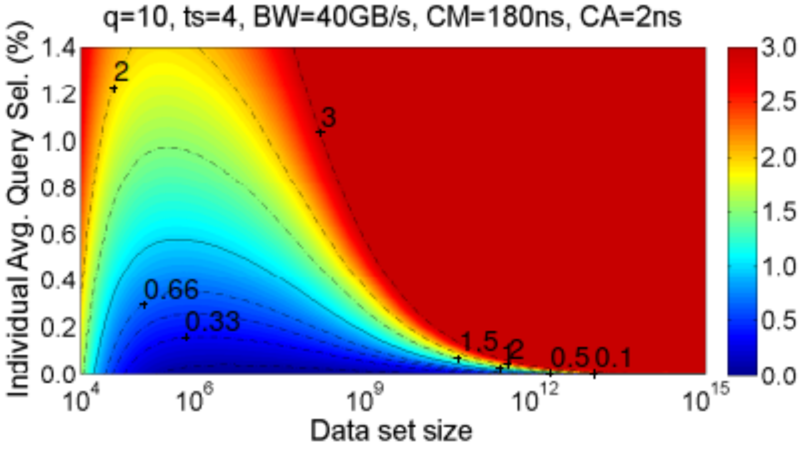
Results as a Function of Data Set Size

Heat map generated has a dependency on the number of concurrent queries

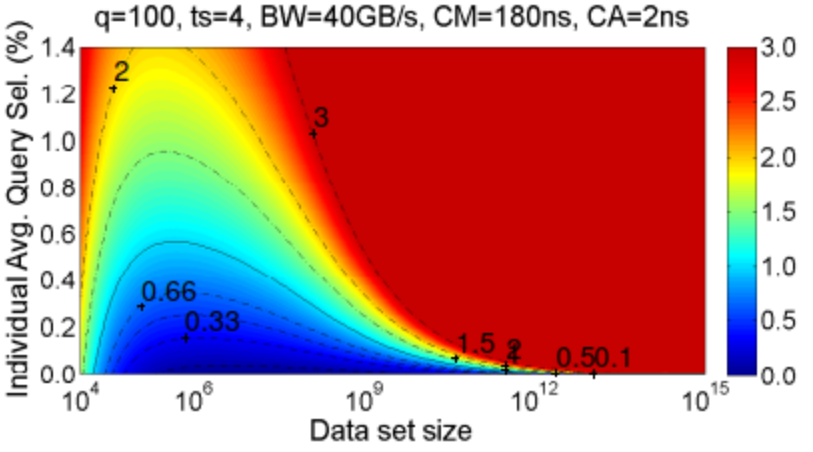
Single Query



Ten Queries

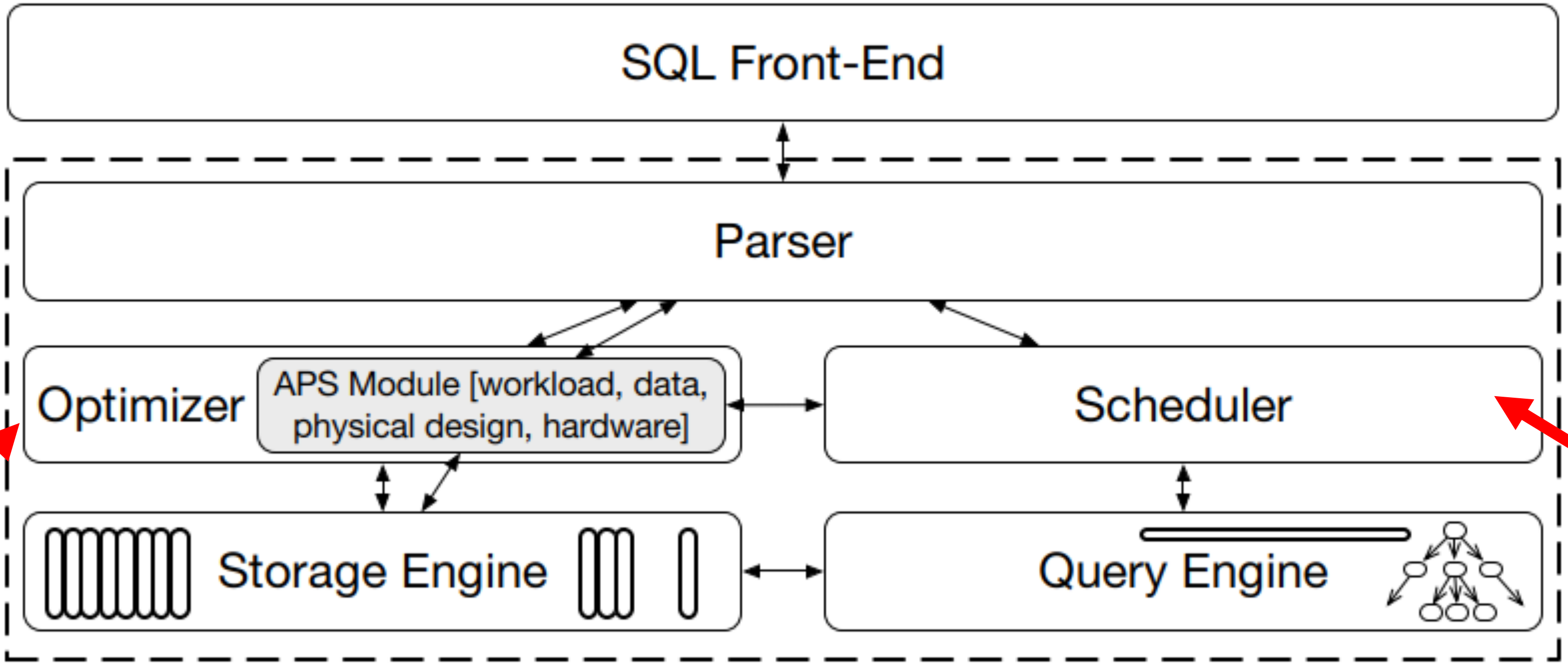


Hundred Queries



Experimental Model

- Access Path Selection Algorithm Implemented

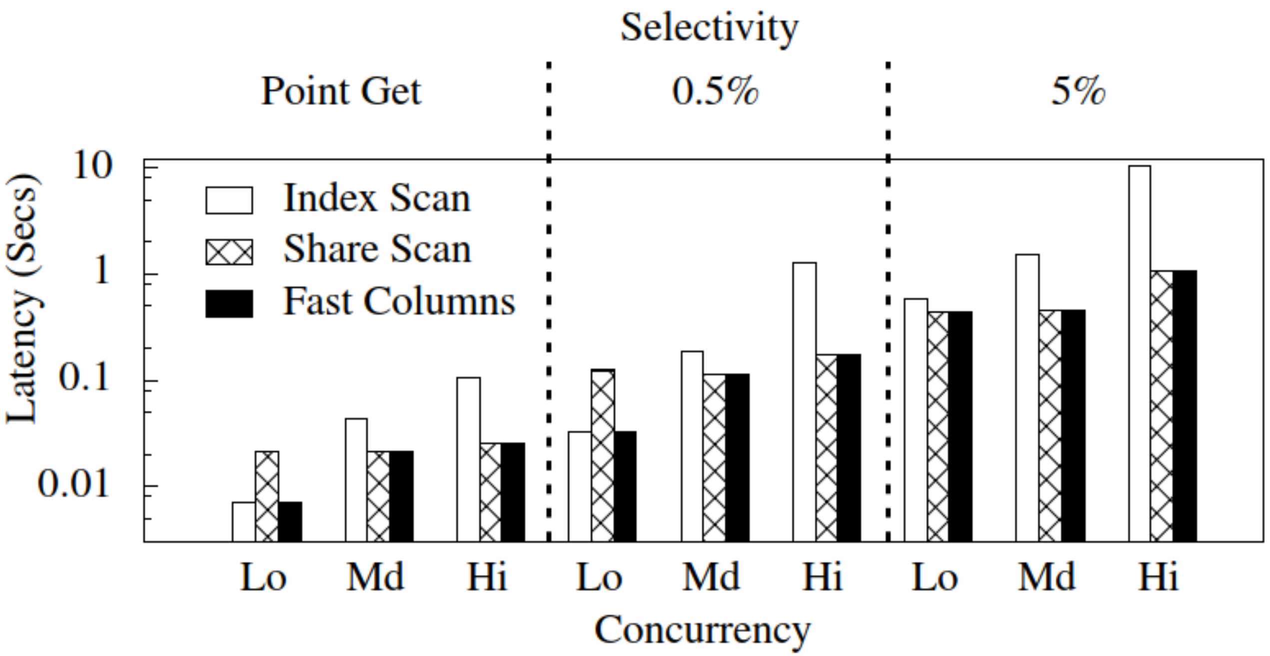


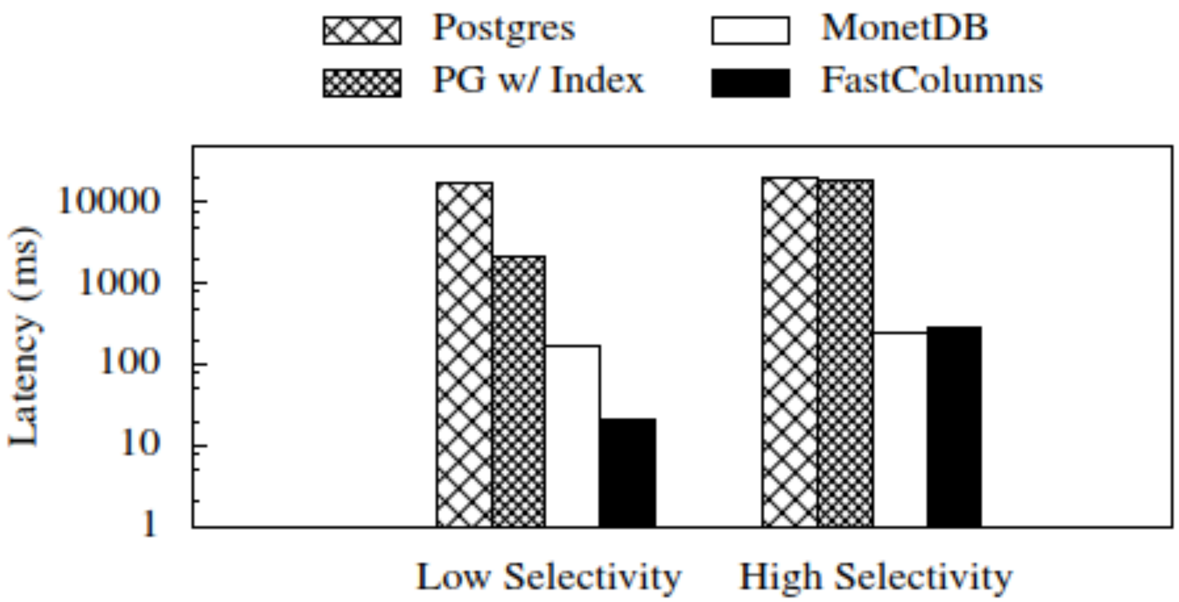
Contains APS algorithm

Helps facilitate optimizer by batching queries

Experimental Model Summary Results

Neither the Index nor Share scans are best throughout but FastColumns is able to make the best of both options





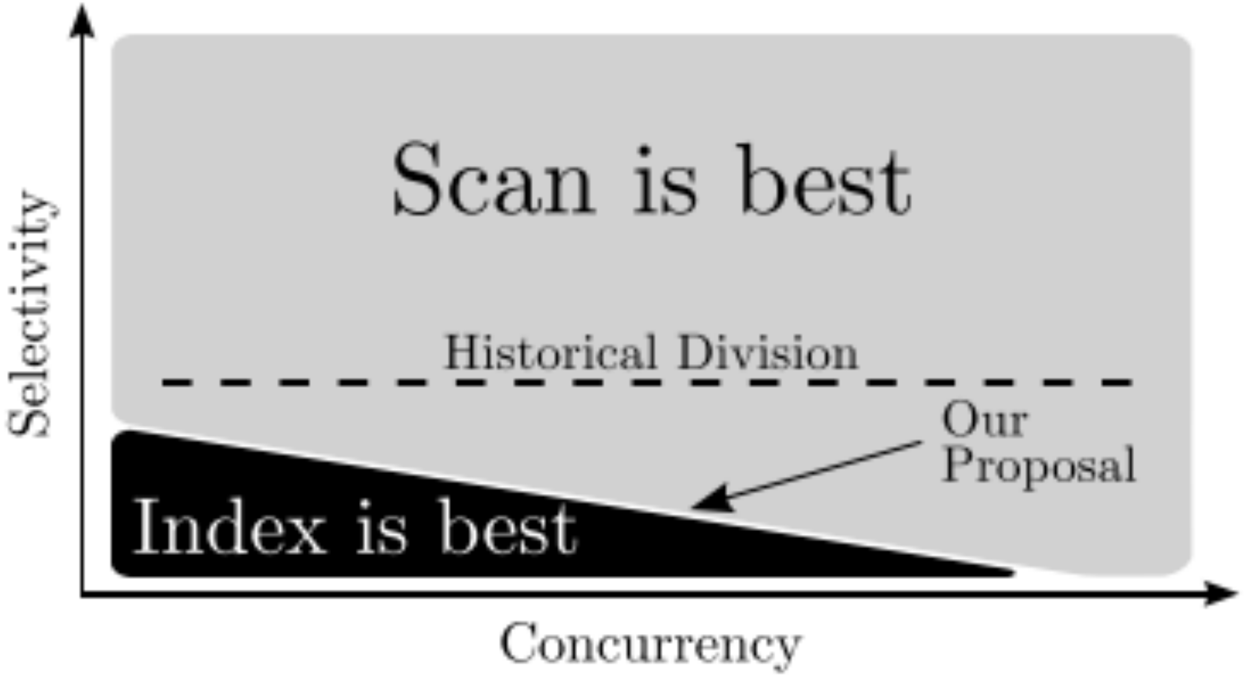
Thoughts on the Paper

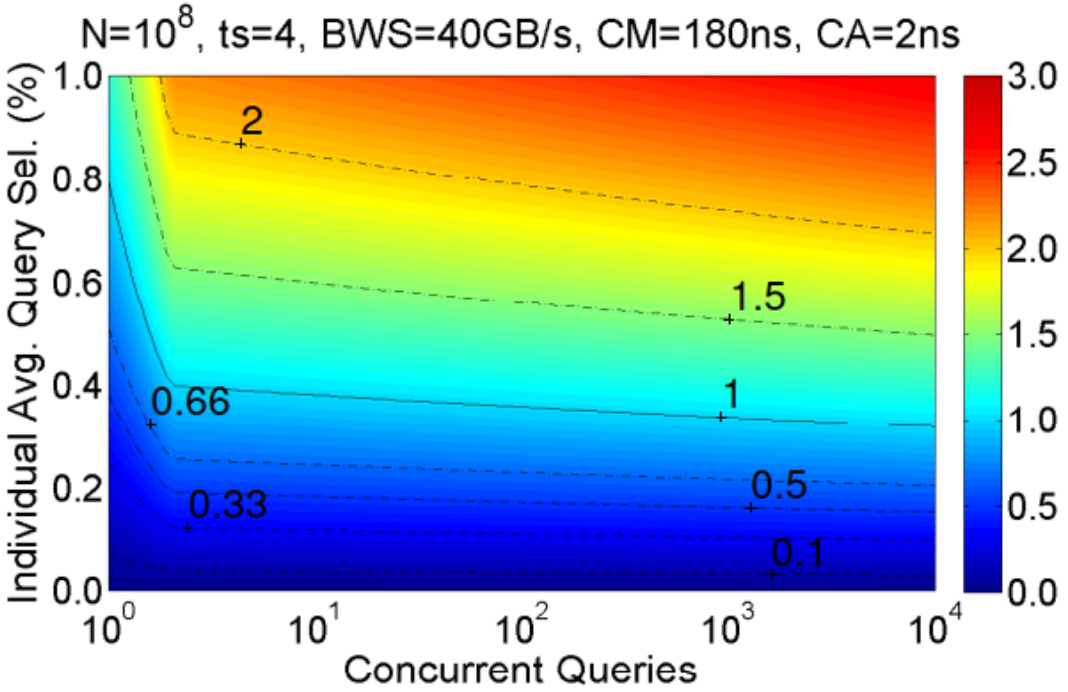
- Brave paper attempting to challenge "common knowledge"
- A lot of thought was put into the paper and there was a really well developed analytical model
    - Analytical model could be used to tune future database systems
    - Would be interesting to dig deeper into the model, maybe with a Monte Carlo

Thoughts on the Paper

- Some information presented in the paper didn’t support the main point the paper tried to convey
  - However, that information showed some more validity to the analytical model
- Disagree with the results and how the paper tried to push the original intent of the study
  - Possible that the results were biased by the hardware selection and additional experimentation with the analytical model would be useful

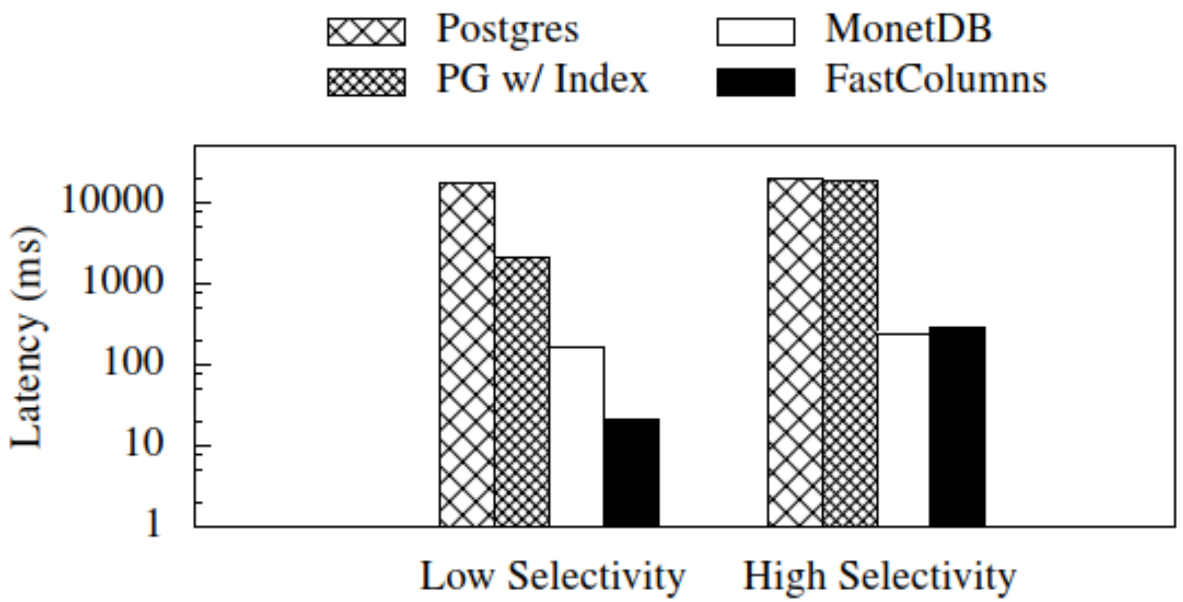
Thoughts on the Paper

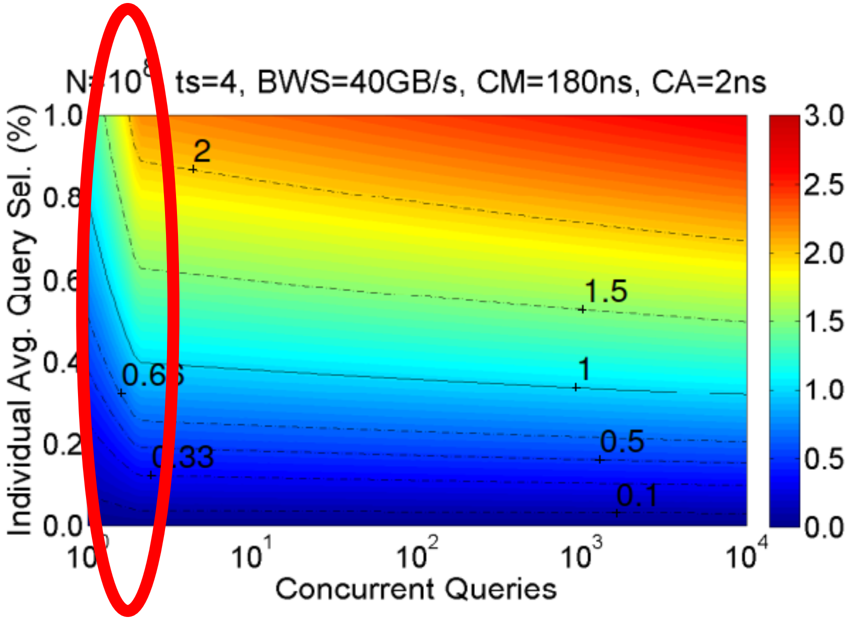




- The index may have some dependency on concurrency but it vanishes really quickly, after a few concurrent queries
  - X Axis is logarithmic

Thoughts on the Paper



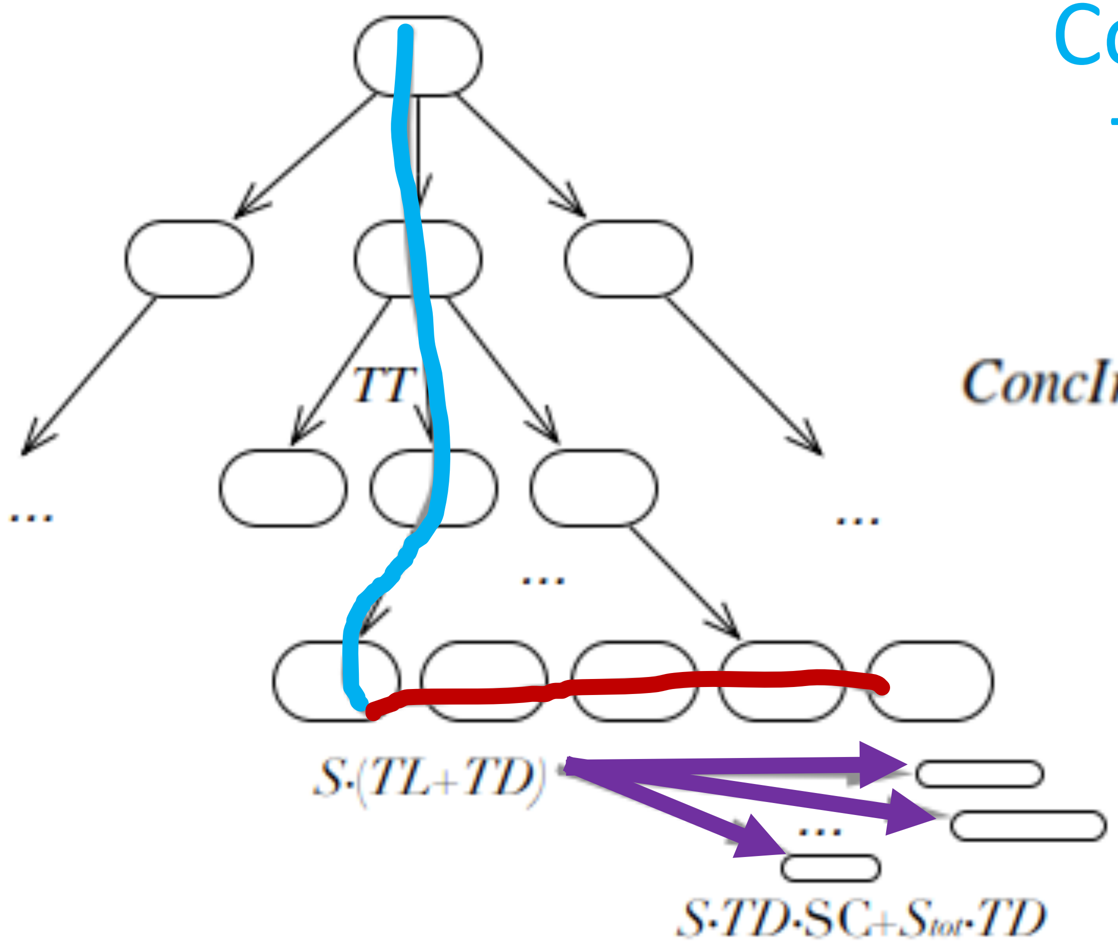


Seems like FastColumns had good results but it’s a single query where the concurrency has large impact

- Monet probably does sequential scan when FastColumns selected an index scan

BACKUP

Index Scan Model



Cost of Tree Traversal

Cost of Writing

$$ConcIndex = q \cdot TT + S_{tot} \cdot (TL + TD_I) + S_{tot} \cdot TD_R + SF \cdot C_A$$

Cost of Leaf and Data Traversal

Cost of Sorting

Index Scan Model

$$ConcIndex = q \cdot TT + S_{tot} \cdot (TL + TD_I) + S_{tot} \cdot TD_R + SF \cdot C_A$$

Workload	q	number of queries
	s_i	selectivity of query i
	S_{tot}	total selectivity of the workload
Dataset	N	data size (tuples per column)
	ts	tuple size (bytes per tuple)
Hardware	C_A	L1 cache access (sec)
	C_M	LLC miss: memory access (sec)
	BW_S	scanning bandwidth (GB/s)
	BW_R	result writing bandwidth (GB/s)
	BW_I	leaf traversal bandwidth (GB/s)
	p	The inverse of CPU frequency
	f_p	Factor accounting for pipelining
Scan & Index	rw	result width (bytes per output tuple)
	b	tree fanout
	aw	attribute width (bytes of the indexed column)
	ow	offset width (bytes of the index column offset)

$$TT = (1 + \lceil log_b(N) \rceil) \cdot \left(C_M + \frac{b \cdot C_A}{2} + \frac{b \cdot f_p \cdot p}{2} \right)$$

$$TL = \frac{N \cdot C_M}{b}$$

$$TD_I = \frac{N \cdot (aw + ow)}{BW_I}$$

$$TD_R = \frac{N \cdot rw}{BW_R}$$

$$SF = S_{tot} \cdot N \cdot log_2(S_{tot} \cdot N)$$

$$SC_i = s_i \cdot N \cdot log_2(s_i \cdot N) \cdot C_A$$

Experimental Model Performance

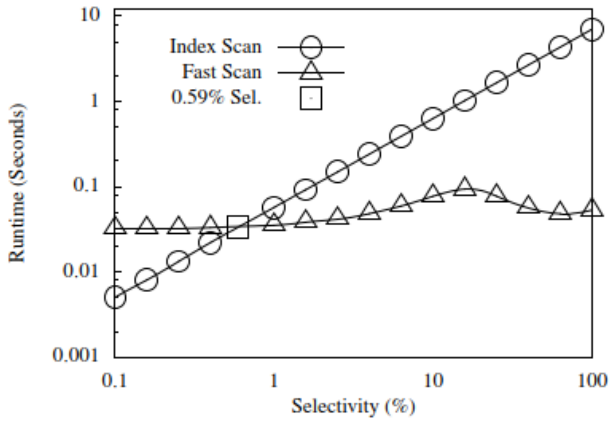


Figure 12: There exists a crossover point for access path selection in analytical systems even when $q = 1$.

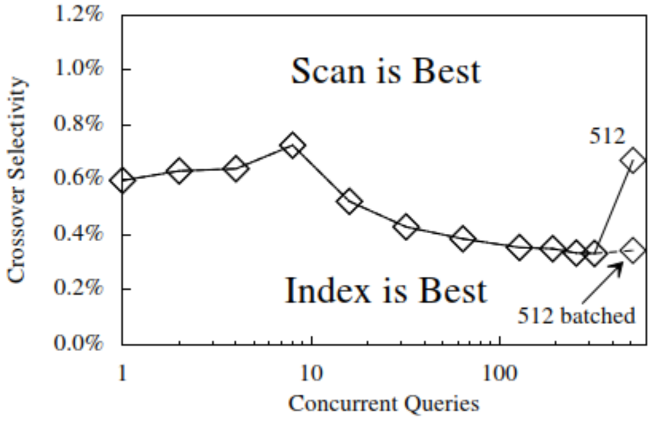


Figure 13: The number of concurrent queries is a critical component of access path selection in analytical data systems.

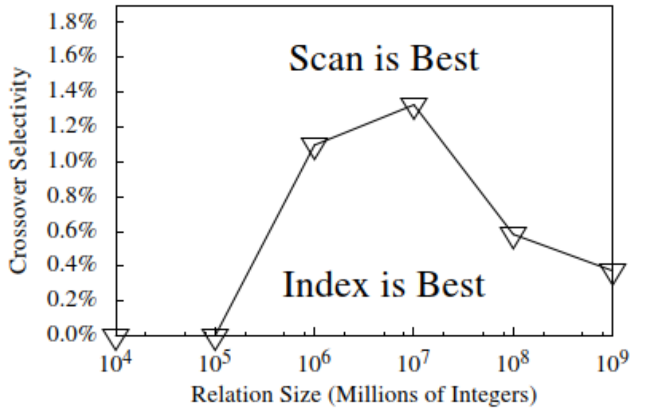


Figure 14: The crossover point is also affected by the data set size ($q = 8$).

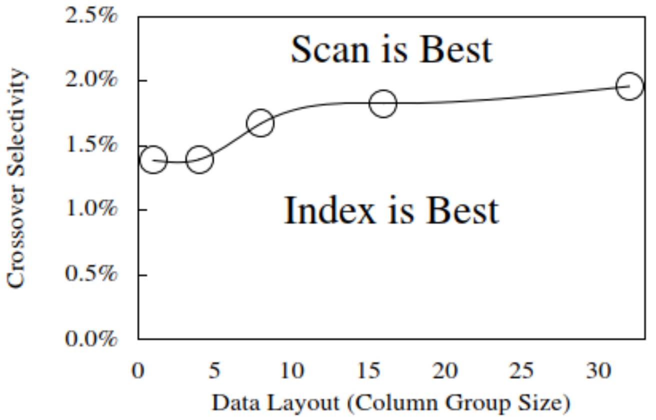


Figure 15: Scans with strided accesses are less efficient increasing the opportunities where an index scan is beneficial.

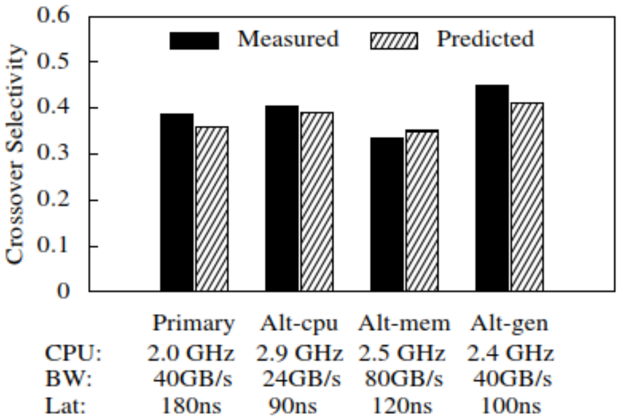


Figure 16: FastColumns is able to accurately predict the crossover point for different hardware configurations.

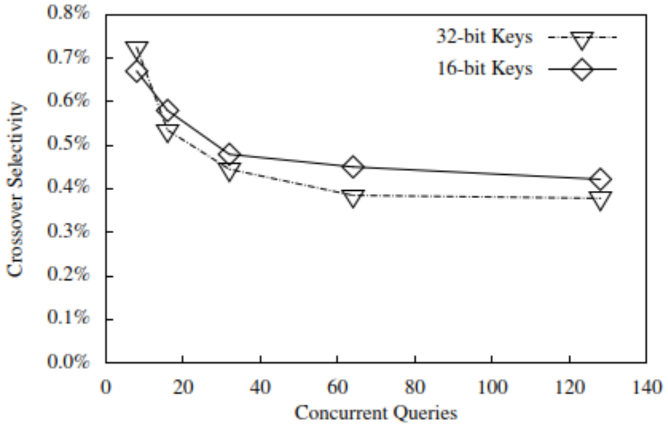


Figure 17: Working directly over compressed data gives a slight advantage for scans.