Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing

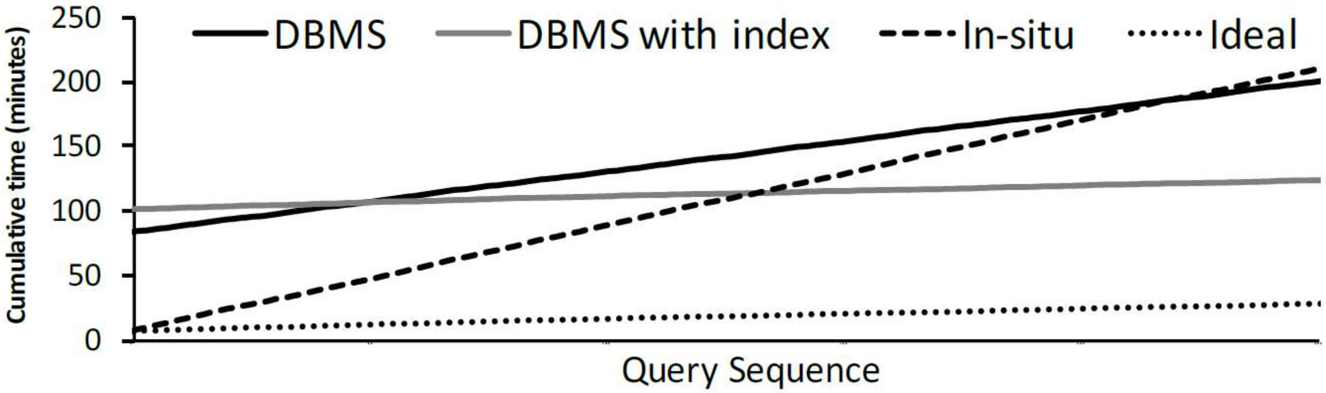
Presented by Ketill Guðmundsson

The problem

- Data generated has grown massively
  - Sensor data
  - Network monitoring data
  - etc
- Current analytical system not built for this much data
  - Data loading is expensive
  - Time-to-insight rising
- Analytical queries are hard to predict
  - No or little workload knowledge

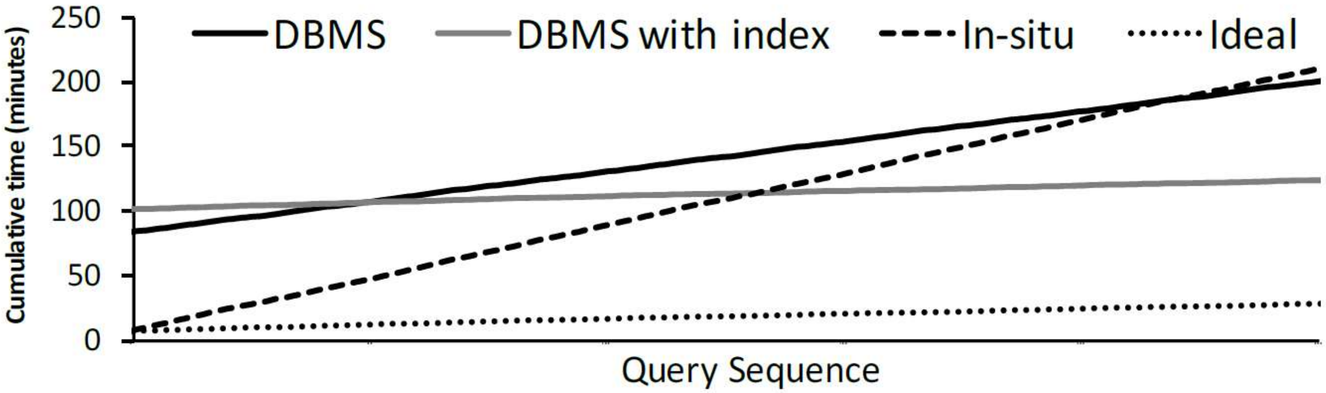
Current solution

- Working over raw data
  - No data loading cost
  - Only logical indexes
  - Physical data is never changed
- In-situ queries



Current solution problems

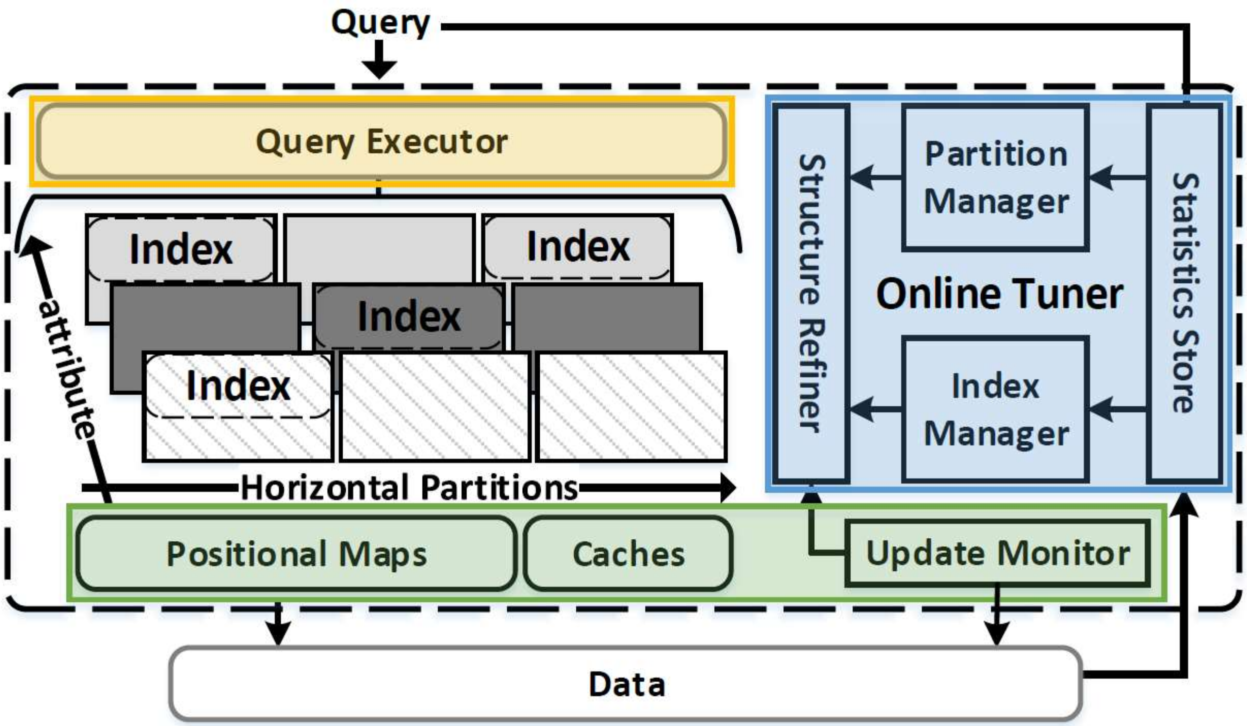
- Using a indexed DBMS overtakes in-situ in time
  - High initial cost
  - Low query cost
- Ideally that never happens



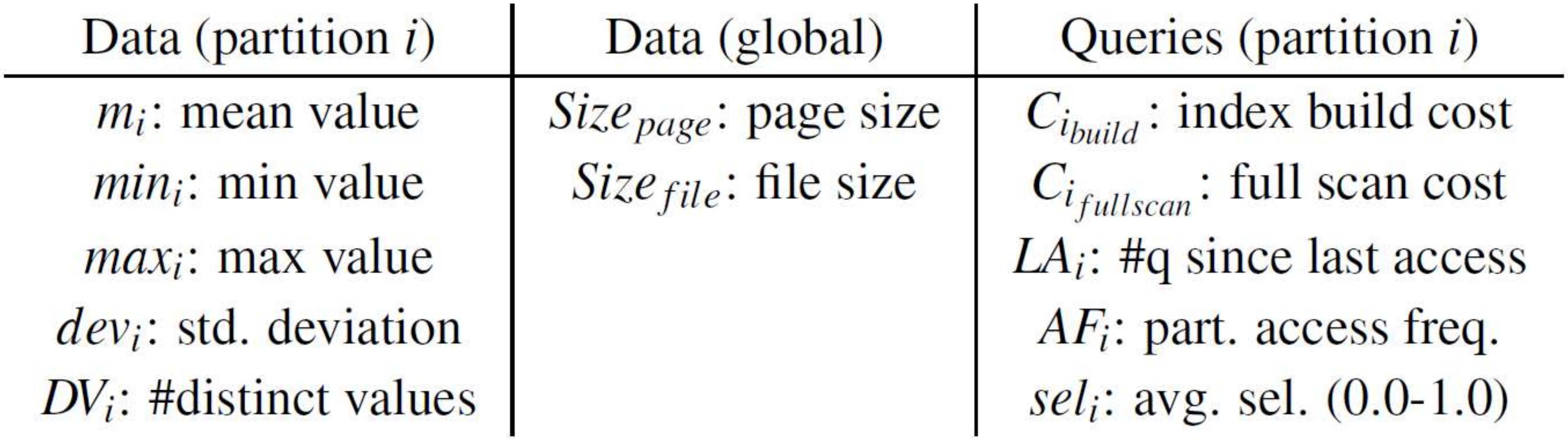
Slalom

- Dynamic partitions
    - Logical partition only
    - Created at runtime
- Dynamic indexes
    - Bloom filters
    - Zonemaps
    - Bitmaps
    - B+ Trees

Slalom architecture



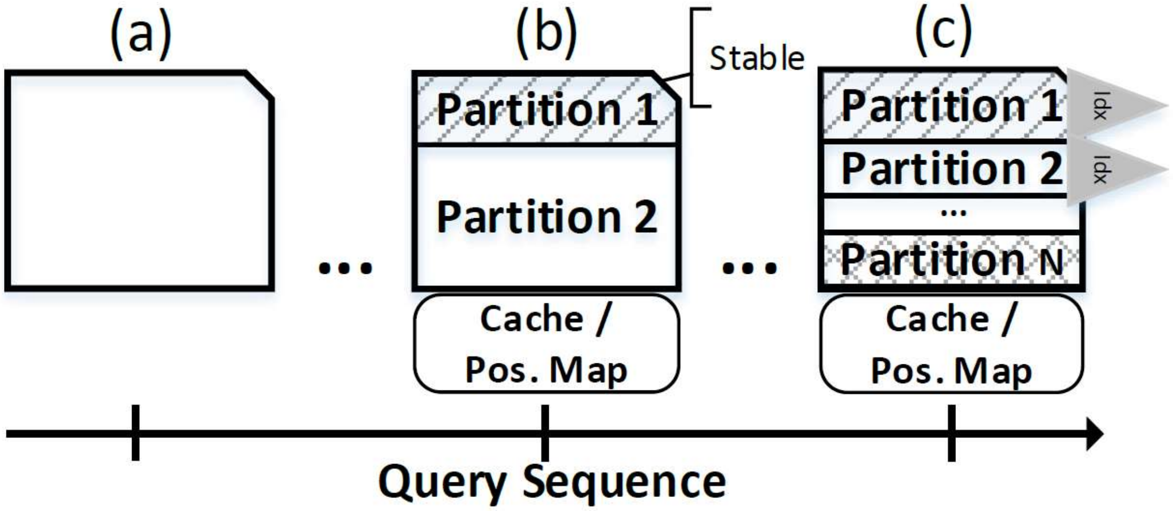
Slalom statistics



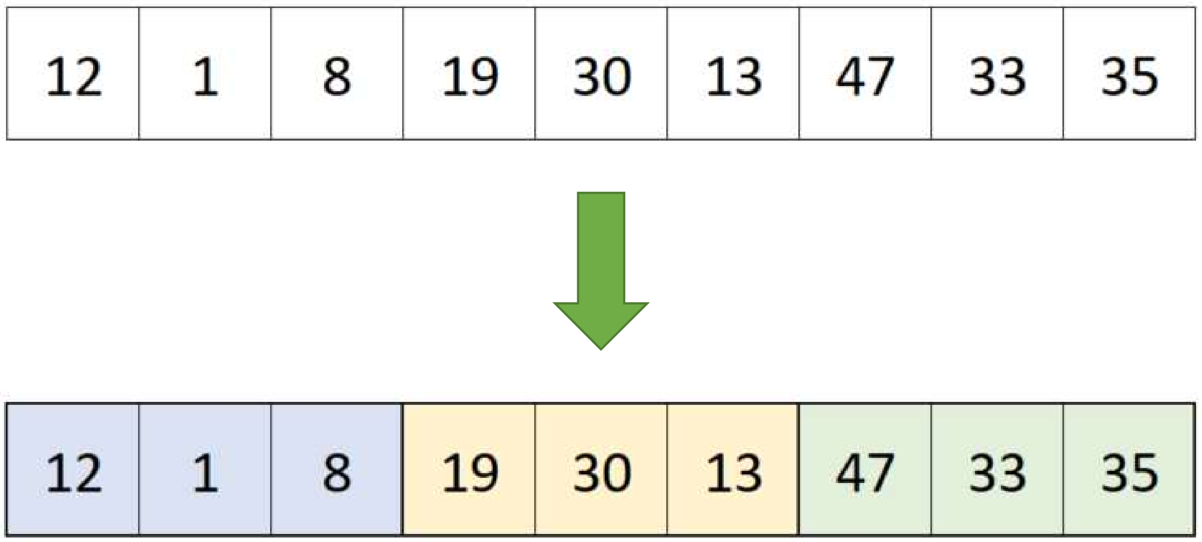
Data (partition i)	Data (global)	Queries (partition i)
m_i: mean value	$Size_{page}$: page size	$C_{i_{build}}$: index build cost
min_i: min value	$Size_{file}$: file size	$C_{i_{fullscan}}$: full scan cost
max_i: max value		LA_i: #q since last access
dev_i: std. deviation		AF_i: part. access freq.
DV_i: #distinct values		sel_i: avg. sel. (0.0-1.0)

Partition manager

- Only logical partitions
- Contiguous and non-overlapping
- Iterative refinement



Partition manager



12 1 8 19 30 13 47 33 35

12 1 8 | 19 30 13 | 47 33 35

Partition manager

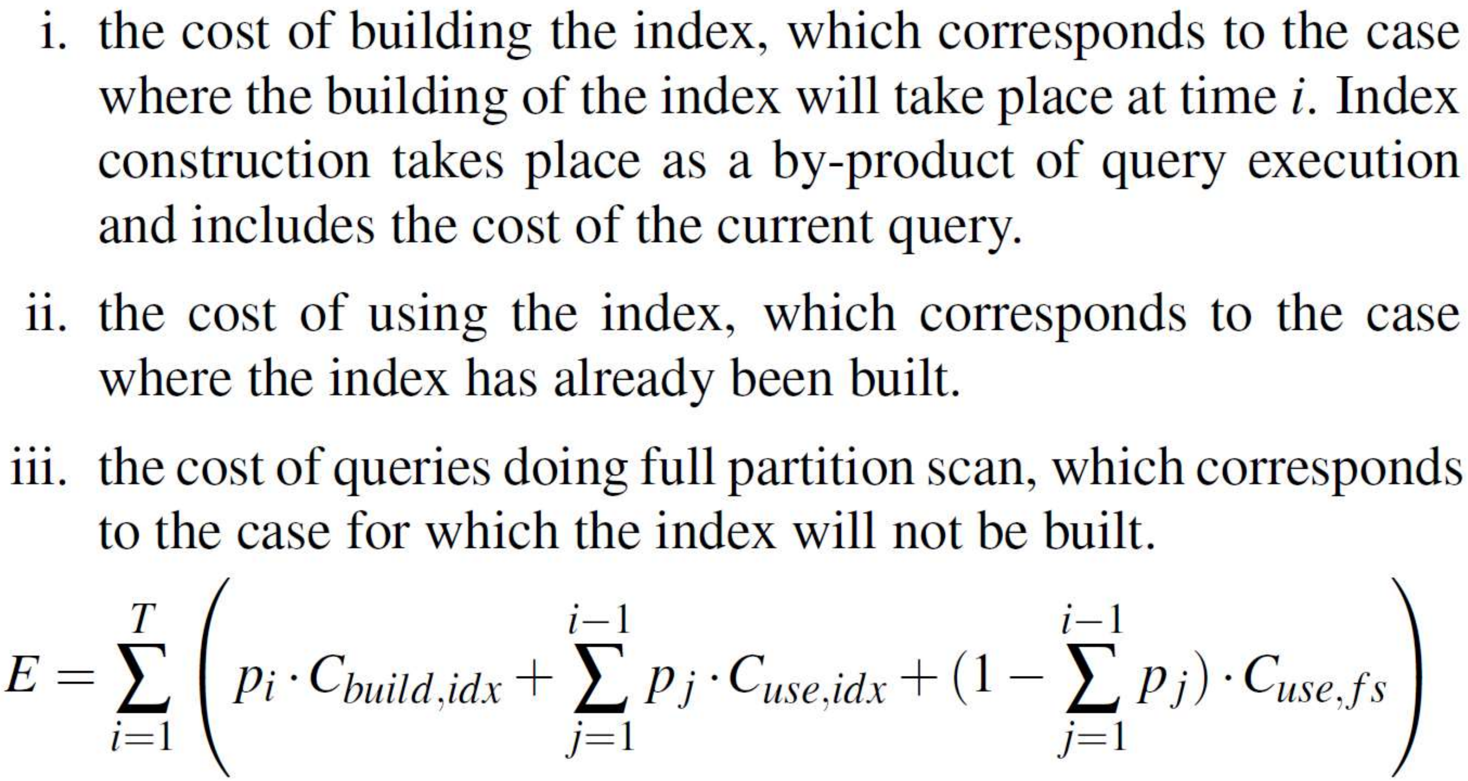
- Incremental splits
- Stops when partition is stable
- Splits into many smaller partitions

$$m = \frac{N \cdot (sel + \log_b(1 - sel))}{\log_b\left(\frac{\sqrt{2 \cdot \pi \cdot sel \cdot N}}{2}\right)} \qquad \text{where} \qquad b = \frac{e}{sel \cdot (1 - sel)}$$

Index manager

- Only applied to stable partitions
- Value existence
    - Bloom filters
    - Zone maps
- Value position
    - B+ Tree

Index manager – Which index?



i. the cost of building the index, which corresponds to the case where the building of the index will take place at time *i*. Index construction takes place as a by-product of query execution and includes the cost of the current query.

ii. the cost of using the index, which corresponds to the case where the index has already been built.

iii. the cost of queries doing full partition scan, which corresponds to the case for which the index will not be built.

$$E = \sum_{i=1}^{T} \left(p_i \cdot C_{build,idx} + \sum_{j=1}^{i-1} p_j \cdot C_{use,idx} + (1 - \sum_{j=1}^{i-1} p_j) \cdot C_{use,fs} \right)$$

Experiments

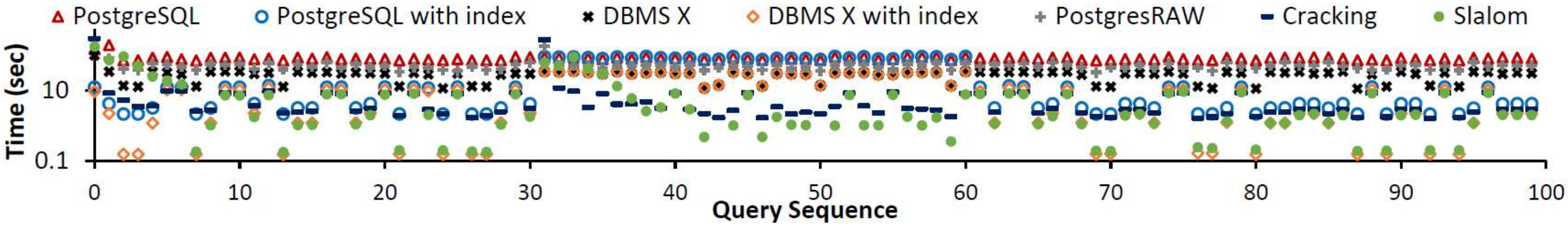


Figure 4: Sequence of 100 queries. Slalom dynamically refines its indexes to reach the performance of an index over loaded data.

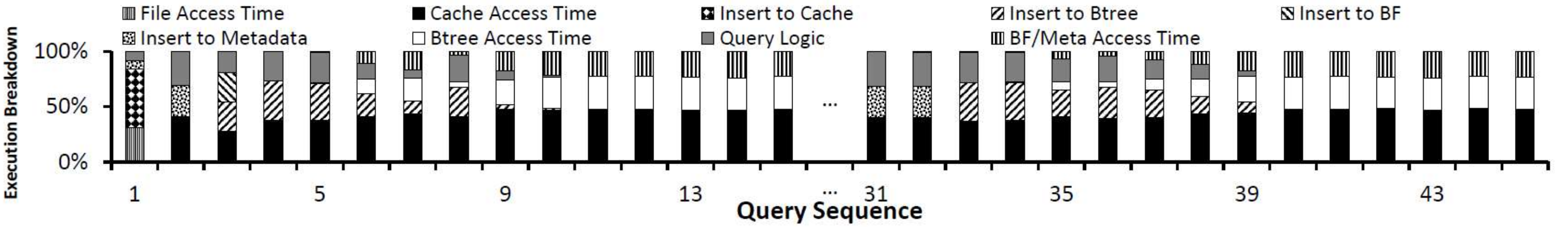


Figure 5: A breakdown of the operations taking place for Slalom during the execution of a subset of the 1000 point query sequence.

Experiments

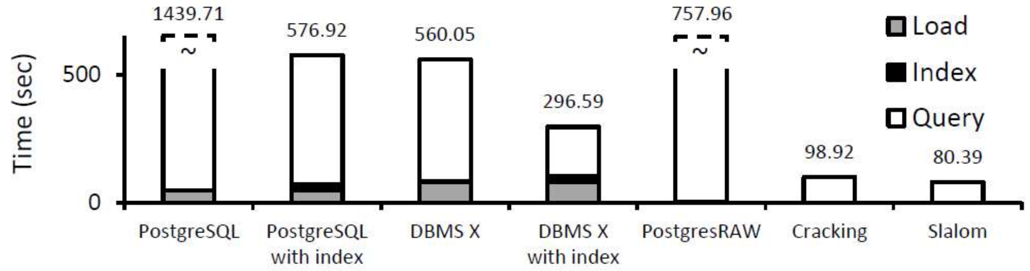


Figure 6: Sequence of 1000 queries. Slalom does not incur loading cost and dynamically builds indexes.

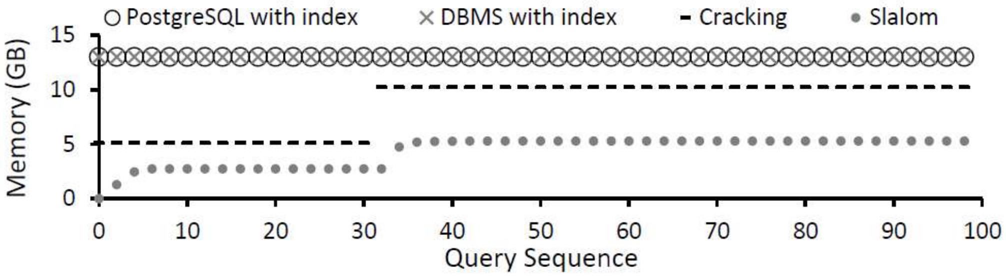


Figure 7: Memory consumption of Slalom vs. a single fully-built B+ Tree for PostgreSQL and DBMS-X. Slalom uses less memory because its indexes only target specific areas of a raw file.

Experiments

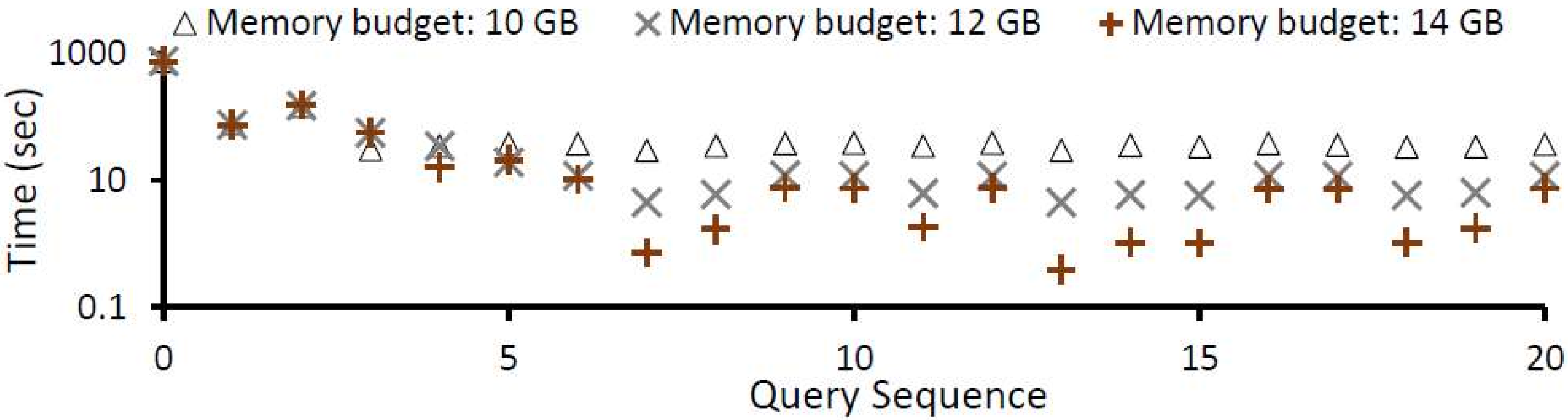
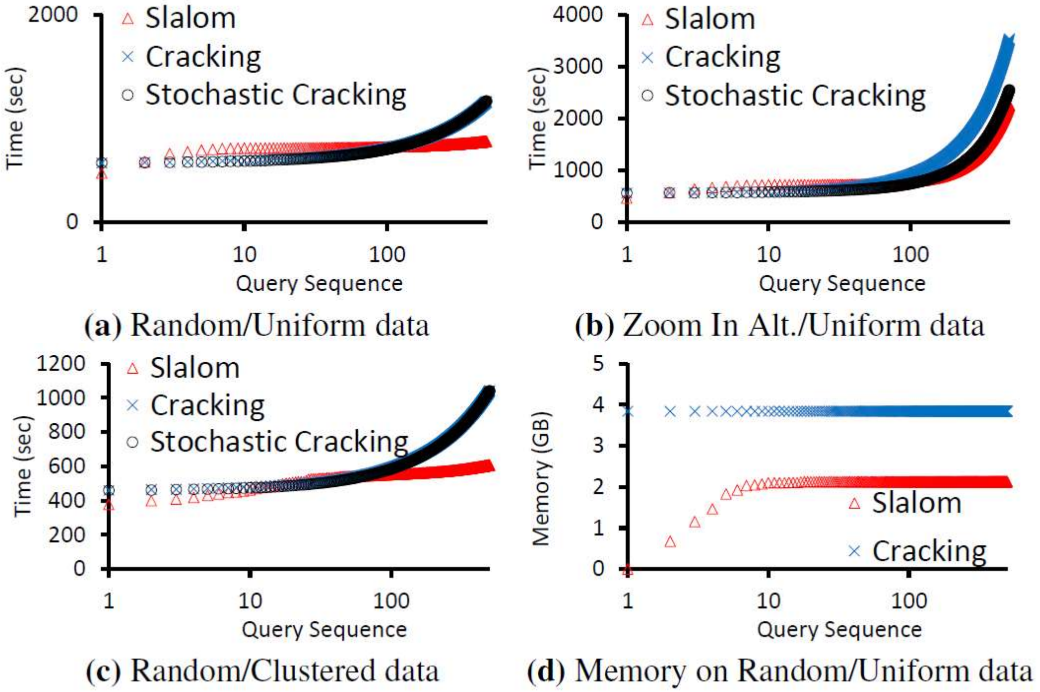


Figure 10: Slalom performance using different memory budgets. Slalom performance varies with alloted memory.

Experiments

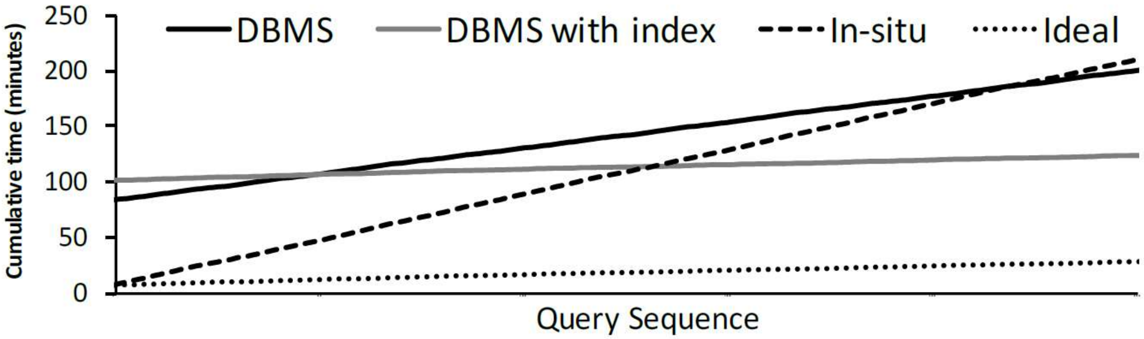


(a) Random/Uniform data (b) Zoom In Alt./Uniform data

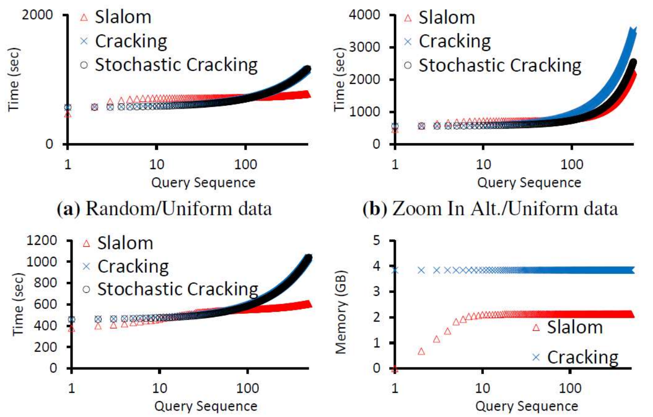
(c) Random/Clustered data (d) Memory on Random/Uniform data

Figure 12: Cracking techniques converge more efficiently but Slalom takes advantage of data distribution.

Conclusion – Positive points



Conclusion – Negative points



(a) Random/Uniform data **(b)** Zoom In Alt./Uniform data

(c) Random/Clustered data **(d)** Memory on Random/Uniform data

Figure 12: Cracking techniques converge more efficiently but Slalom takes advantage of data distribution.