Wave-Indices: Indexing Evolving Databases

Hector Garcia-Molina – Stanford University
Narayanan Shivakumar – Stanford University

Presentation By: Mirza Beg

Outline

- Problem Statement
- Formalisms
- Proposed Techniques
- Analytic Comparison
- Case Studies
- Conclusion
- Discussion

Problem Statement

- Large amounts of data being generated everyday.
- Need to index into the data of some past window of days:
  - Search engines – provide an index for the past 30 days of some news articles.
  - Financial institutions – keep an index of the stock market trades for the past 7 days
- Each day a new batch of data must be added to the index, and data older than the window should be removed.
- Where would we need this?
  - Application semantics require sliding window – Streams
  - Users interest wanes over time – News
  - Want to reduce storage costs – Too much data -> slow access

Sliding Window Indices

- Have been in use for many years
- Need to revisit existing schemes because of the increasing volumes of data
  - Web Search – Engines need to keep track of ever-increasing web pages, articles and other information.
  - Data Warehousing – huge volumes of sales, banking and other transactions -> need efficient access to information
  - SCAM (Stanford Copy Analysis Mechanism) – detecting illegal copies of copyrighted digital documents -> interested only in documents collected over the past few weeks.
- Solution ?
  - Wave Indices

What are Wave Indices ?

- **Traditional Indexing**
  - Keep a single conventional index, and every day delete the old batch of data and insert the new batch of data into it.

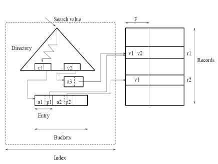
    I {d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}}

    Operation update - add d_{11} and delete d_1

    I {d_{11}, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}}
- **Wave Index**
  - Index a window of W days and partition the data across multiple indices.
  - Create W/n indices where n is the number of constituent indices.

    I_1 {d_1, d_2, d_3} , I_2 { d_4, d_5, d_6} , I_3 { d_7, d_8, d_9, d_{10}}
  - Service queries by accessing all indices.
  - Update operations and techniques ?

Index Structures



- Data we need to index exists as *records* r_1, r_2.
- Each of the records has a *search field* F.
  - Index is built on the search field.
  - Each record may have multiple values for F.
  - Thus multiple buckets may be referencing the same record.
- Index consists of a *directory* and associated *buckets*.
  - Directory is a search structure
  - Given a value v identifies a bucket *b*
- Each bucket *b* has a pointer p_i to record r_i and may have additional information a_i (timestamp)
- Queries can possibly scan the whole index (aggregate functions).

*directory is in memory and buckets are stored contiguously on disk.

Index Update Techniques

- **In-Place Updating**
  - The directory and/or buckets are modified in place.
  - If bucket is full, then bucket is copied to a new location and allocated more space.
  - Resulting index is not packed even if the original was packed.
  - Concurrency control required during updates.
- **Simple Shadow Updating**
  - Makes a copy of the index
  - For each update modifies the new copy of the index in place
  - The new index replaces the old version in the wave index
  - No concurrency control required.
- **Packed Shadow Updating**
  - Same as Simple Shadow except that the resulting index is packed.

*packed – an index is said to be packed if each of its buckets are allocated contiguously on disk and use the minimal amount of space to store entries.

Operations in Building Wave Indices

- Primitive Functions on a Wave Index θ.
  - **AddIndex** (I, θ) – Adds I to the set of constituent indices in θ.
  - **DropIndex** (I, θ) – Removes I form θ. Deletes all entries in I.
  - **BuildIndex** (Days, I) – Builds a packed index I for the batch of records in those days.
  - **AddToIndex** (Days, I) – Incrementally adds the batch of entries for Days records to I.
  - **DeleteFromIndex** (Days, I) – Incrementally deletes entries for Days from I.
  - **TimedIndexProbe** (θ, T_1, T_2, s) – For a wave index θ, T_1 and T_2 retrieves buckets of entries for s inserted between T_1 and T_2.
  - **TimedSegmentScan** (θ, T_1, T_2) – Retrieves all entries inserted between T_1 and T_2.

DEL - Deletion based

- Initially index W/n days of data each in indices $I_1, \ldots, I_n$.
- Then make $I_1, \ldots, I_n$ constituent indexes of wave Index θ.
- When d_{new} is available, delete entries of $d_{new\text{-}W}$ from I_j that indexed $d_{new\text{-}W}$.
- Then we insert entries for d_{new} to I_j.

Day	New Data	Operation	Index File (I_1)	Index File (I_2)
10	$+ d_1, \ldots, d_{10}$	Start	$\{ d_1, d_2, d_3, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
11	$+ d_{11}$	Delete d_1 from I_1	$\{ d_2, d_3, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
		Add d_{11} to I_1	$\{ d_{11}, d_2, d_3, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
12	$+ d_{12}$	Delete d_2 from I_1	$\{ d_{11}, d_3, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
		Add d_{12} to I_1	$\{ d_{11}, d_{12}, d_3, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
13	$+ d_{13}$	Delete d_3 from I_1	$\{ d_{11}, d_{12}, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$
		Add d_{13} to I_1	$\{ d_{11}, d_{12}, d_{13}, d_4, d_5 \}$	$\{ d_6, d_7, d_8, d_9, d_{10} \}$

Table 1: Deletion based index maintenance ($W = 10$).

- When d_{11} is available on day 11, first delete d_1 from I_1.
- Index d_{11} into I_1.
- DEL maintains hard windows.

REINDEX - Reindexing

- Initially index W/n days of data each in indices $I_1, \ldots, I_n$.
- Then make $I_1, \ldots, I_n$ constituent indexes of wave Index θ.
- When d_{new} is available, delete all entries I_j that indexed d_{new-W} .
- Then we rebuild I_j with entries for $d_{new-W}, \ldots, d_{new}$.

Day	New Data	Operation	Index File I_1	Index File I_2
10	$+ d_1, \ldots, d_{10}$	Start	$\{d_1, d_2, d_3, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
11	$+d_{11}$	Reindex $d_2, d_3, d_4, d_5, d_{11}$	$\{d_2, d_3, d_4, d_5, d_{11}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
12	$+d_{12}$	Reindex $d_3, d_4, d_5, d_{11}, d_{12}$	$\{d_3, d_4, d_5, d_{11}, d_{12}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
13	$+d_{13}$	Reindex $d_4, d_5, d_{11}, d_{12}, d_{13}$	$\{d_4, d_5, d_{11}, d_{12}, d_{13}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
14	$+d_{14}$	Reindex $d_5, d_{11}, d_{12}, d_{13}, d_{14}$	$\{d_5, d_{11}, d_{12}, d_{13}, d_{14}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
15	$+d_{15}$	Reindex $d_{11}, d_{12}, d_{13}\ d_{14}\ d_{15}$	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$
16	$+d_{16}$	Reindex $d_7, d_8, d_9, d_{10}, d_6$	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_7, d_8, d_9, d_{10} d_{16}\}$

Table 2: Reindexing based index maintenance ($W = 10, n = 2$).

- When data d_{11} is available replace d_1 in I_1 by d_{11} by rebuilding index I_1 with data d_2, d_3, d_4, d_5 and d_{11}. Similarly with subsequent days.
- Maintains hard windows.
- Requires reindexing W/n days of data every day.

REINDEX$^+$ - REINDEX Enhanced

- REINDEX$^+$ maintains a temporary index, *Temp*.
- Avoids re-computing index entries everyday.

Day	Operation	Index I_1	Index I_2	Temp
10	Temp $\leftarrow \phi$ $I_1 \leftarrow$ BuildIndex($\{1,2,3,4,5\}$) $I_2 \leftarrow$ BuildIndex($\{6,7,8,9,10\}$)	$\{d_1, d_2, d_3, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	ϕ
11	*Temp*, $I_1 \leftarrow$ BuildIndex($\{11\}$) AddToIndex($\{2,3,4,5\}, I_1$)	$\{d_{11}, d_2, d_3, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$\{d_{11}\}$
12	AddToIndex($\{12\}$, Temp) $I_1 \leftarrow$ Temp AddToIndex($\{3,4,5\}, I_1$)	$\{d_{11}, d_{12}, d_3, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$\{d_{11}, d_{12}\}$
13	AddToIndex($\{13\}$, Temp) $I_1 \leftarrow$ Temp AddToIndex($\{4,5\}, I_1$)	$\{d_{11}, d_{12}, d_{13}, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$\{d_{11}, d_{12}, d_{13}\}$
14	AddToIndex($\{14\}$, Temp) $I_1 \leftarrow$ Temp AddToIndex($\{5\}, I_1$)	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$\{d_{11}, d_{12}, d_{13}, d_{14}\}$
15	$I_1 \leftarrow$ Temp AddToIndex($\{15\}, I_1$) Temp $\leftarrow \phi$	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	ϕ
16	*Temp*, $I_2 \leftarrow$ BuildIndex($\{16\}$) AddToIndex($\{7,8,9,10\}, I_2$)	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_{16}, d_7, d_8, d_9, d_{10}\}$	$\{d_{16}\}$

Table 5: Example of index Transitions in *REINDEX*$^+$ (W = 10, n = 2).

- On average reindexes about half the number of days REINDEX does.

REINDEX^{++} - Enhanced REINDEX$^+$

- Maintains more than one Temporary indices $\{ T_1, \ldots, T_n \}$.
- Performs most of the maintenance work for the wave index in advance.

Day	Operation	Index I_1	Index I_2	Temp
10	$I_1 \leftarrow$ BuildIndex($\{1,2,3,4,5\}$) $I_2 \leftarrow$ BuildIndex($\{6,7,8,9,10\}$) $T_0 \leftarrow \phi$, $T_1 \leftarrow$ BuildIndex($\{5\}$) $T_2 \leftarrow T_1$, AddToIndex($\{4\}, T_2$) $T_3 \leftarrow T_2$, AddToIndex($\{3\}, T_3$) $T_4 \leftarrow T_3$, AddToIndex($\{2\}, T_4$)	$\{d_1, d_2, d_3, d_4, d_5\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_0 = \phi$, $T_1 = \{d_5\}$ $T_2 = \{d_5, d_4\}$ $T_3 = \{d_5, d_4, d_3\}$ $T_4 = \{d_5, d_4, d_3, d_2\}$
11	AddToIndex($\{11\}, T_4$) Rename T_4 as I_1 AddToIndex($\{11\}, T_3$)	$\{d_5, d_4, d_3, d_2, d_{11}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_4 = \{d_5, d_4, d_3, d_2, d_{11}\}$ $T_3 = \{d_5, d_4, d_3, d_{11}\}$
12	AddToIndex($\{12\}, T_3$) Rename T_3 as I_1 AddToIndex($\{11,12\}, T_2$)	$\{d_5, d_4, d_3, d_{11}, d_{12}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_3 = \{d_5, d_4, d_3, d_{11}, d_{12}\}$ $T_2 = \{d_5, d_4, d_{11}, d_{12}\}$
13	AddToIndex($\{13\}, T_2$) Rename T_2 as I_1 AddToIndex($\{11,12,13\}, T_1$)	$\{d_5, d_4, d_{11}, d_{12}, d_{13}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_2 = \{d_5, d_4, d_{11}, d_{12}, d_{13}\}$ $T_1 = \{d_5, d_{11}, d_{12}, d_{13}\}$
14	AddToIndex($\{14\}, T_1$) Rename T_1 as I_1 AddToIndex($\{11,12,13,14\}, T_0$)	$\{d_5, d_{11}, d_{12}, d_{13}, d_{14}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_1 = \{d_5, d_{11}, d_{12}, d_{13}, d_{14}\}$ $T_0 = \{d_{11}, d_{12}, d_{13}, d_{14}\}$
15	AddToIndex($\{15\}, T_0$) Rename T_0 as I_1 $T_0 \leftarrow \phi$, $T_1 \leftarrow$ BuildIndex($\{10\}$) $T_2 \leftarrow T_1$, AddToIndex($\{9\}, T_2$) $T_3 \leftarrow T_2$, AddToIndex($\{8\}, T_3$) $T_4 \leftarrow T_3$, AddToIndex($\{7\}, T_4$)	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_6, d_7, d_8, d_9, d_{10}\}$	$T_0 = \{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$ $T_0 = \phi$, $T_1 = \{d_{10}\}$ $T_2 = \{d_{10}, d_9\}$ $T_3 = \{d_{10}, d_9, d_8\}$ $T_4 = \{d_{10}, d_9, d_8, d_7\}$
16	AddToIndex($\{16\}, T_4$) Rename T_4 as I_2 AddToIndex($\{16\}, T_3$)	$\{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$	$\{d_{10}, d_9, d_8, d_7, d_{16}\}$	$T_4 = \{d_{11}, d_{12}, d_{13}, d_{14}, d_{15}\}$ $T_3 = \{d_{10}, d_9, d_8, d_{16}\}$

Table 6: Example of index transitions in *REINDEX*$^{++}$ (W = 10, n = 2).

- Increased storage requirements at the cost of making data available sooner.
- Does about the same amount of work as REINDEX+ but reduces time to index a new day's data.

WATA - Wait And Throw Away

- Uses lazy form of deletion by throwing away an entire index when all its entries have expired.
- When new data is available, add it to index with unused capacity.
- When no such index available, first throw away Index with the oldest data.

Day	New Data	Operation	Index File I_1	Index File I_2	Index File I_3	Index File I_4
10	$+d_1, \ldots, d_{10}$	Start	$\{ d_1, d_2, d_3 \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10} \}$
11	$+ d_{11}$	Add d_{11} to I_4	$\{ d_1, d_2, d_3 \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11} \}$
12	$+ d_{12}$	Add d_{12} to I_4	$\{ d_1, d_2, d_3 \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11}, d_{12} \}$
13	$+ d_{13}$	Drop I_1	-	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11}, d_{12} \}$
		Create $I_1 = \phi$	$\{ \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11}, d_{12} \}$
		Add d_{13} to I_1	$\{ d_{13} \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11}, d_{12} \}$
14	$+ d_{14}$	Add d_{14} to I_1	$\{ d_{13}, d_{14} \}$	$\{ d_4, d_5, d_6 \}$	$\{ d_7, d_8, d_9 \}$	$\{ d_{10}, d_{11}, d_{12} \}$

- For the first 10 days index data into $I_1, \ldots, I_4$
- When data d_{11} is available, add it to I_4. Similarly for d_{12}. When data d_{13} is available on the 13th day, first throw away I_1. then create a new index I_1, and finally add d_{13} to it. The next day add d_{14} to I_1, and so on.
- Maintains soft windows -> Uses more space -> Relatively little work each day.
- Requires at least two constituent indices.

RATA - Reindex And Throw Away

- A hybrid of REINDEX^{++} and WATA.
- Uses temporary indices T_i to be computed in advance to replace some I_j.

State	Operation	Index I_1	Index I_2	Index I_3	Index I_4	Temp
10	$I_1 \leftarrow$ BuildIndex($\{1,2,3\}$) $I_2 \leftarrow$ BuildIndex($\{4,5,6\}$) $I_3 \leftarrow$ BuildIndex($\{7,8,9\}$) $I_4 \leftarrow$ BuildIndex($\{10\}$) $T_0 \leftarrow$ BuildIndex($\{3\}$) $T_1 \leftarrow T_0$, AddToIndex($\{2\}, T_0$)	$\{d_1, d_2, d_3\}$	$\{d_4, d_5, d_6\}$	$\{d_7, d_8, d_9\}$	$\{d_{10}\}$	$T_0 = \{d_3\}$ $T_1 = \{d_3, d_2\}$
11	AddToIndex($\{11\}, I_4$) Drop I_1, Rename T_0 as I_1	$\{d_3, d_2\}$	$\{d_4, d_5, d_6\}$	$\{d_7, d_8, d_9\}$	$\{d_{10}, d_{11}\}$	Unchanged: T_1
12	AddToIndex($\{12\}, I_4$) Drop I_1, Rename T_1 as I_1	$\{d_3\}$	$\{d_4, d_5, d_6\}$	$\{d_7, d_8, d_9\}$	$\{d_{10}, d_{11}, d_{12}\}$	
13	$I_1 \leftarrow$ BuildIndex($\{13\}$) $T_0 \leftarrow$ BuildIndex($\{6\}$) $T_1 \leftarrow T_0$, AddToIndex($\{5\}, T_1$)	$\{d_{13}\}$	$\{d_4, d_5, d_6\}$	$\{d_7, d_8, d_9\}$	$\{d_{10}, d_{11}, d_{12}\}$	$T_0 = \{d_6\}$ $T_1 = \{d_6, d_5\}$
14	AddToIndex($\{14\}, I_1$) Drop I_2, Rename T_0 as I_2	$\{d_{13}, d_{14}\}$	$\{d_6, d_5\}$	$\{d_7, d_8, d_9\}$	$\{d_{10}, d_{11}, d_{12}\}$	Unchanged: T_1

Table 7: Example of index transitions in RATA

- For the first 10 days index data into $I_1, \ldots, I_4$. Builds temp indices T_0, T_1.
- Indexes $\{d_3\}$ -> T0 and $\{d_3, d_2\}$ -> T. Later drops I_1 and replace I_1 with T_0.
- Performs more work than WATA. Maintains hard windows.
- Takes the same time as WATA to make the data available.

Analytic Comparison

Measure/ Scheme	Max Space [Operation]	Avg Space [Operation]	Max Space [Transition]	Avg Space [Transition]
DEL	$W * S'$	$W * S'$	$\lceil X \rceil * S'$	$X * S'$
REINDEX	$W * S$	$W * S$	$\lceil X \rceil * S$	$X * S$
REINDEX$^+$	$(W + \lceil X \rceil) * S'$	$(W + \frac{X}{2}) * S'$	$\lceil X \rceil * S'$	$\frac{X}{2} * S'$
REINDEX^{++}	$(W + \frac{1}{2} * \lceil X \rceil * \lceil X - 1 \rceil) * S'$	$(W + \frac{X^2}{6} + \frac{X}{2}) * S'$	0	0
WATA*	$(W + \lceil Y - 1 \rceil) * S'$	$(W + \frac{Y}{2}) * S'$	$\lceil Y \rceil * S'$	$\frac{Y}{2} * S'$
RATA*	$(W + \frac{1}{2} * \lceil Y \rceil * \lceil Y - 1 \rceil) * S'$	$(W + \frac{Y^2}{6} + \frac{Y}{2}) * S'$	$\lceil Y \rceil * S'$	$\frac{Y}{2} * S'$

Table 8: Space utilization of wave indexes that use simple shadow updating ($X = \frac{W}{n}, Y = \frac{W-1}{n-1}$).

- Working with a window of W days.
- S is the space required to store a packed index of one day.
- S' is the space required to store a non-packed index of one day.
- REINDEX$^+$ requires an average of (W+X/2) * S' and a maximum of (W+[X]-1) * S' when it indexes constituent indices for W days. REINDEX$^+$ *Temp* indexes at most [X-1] days. However when averaged is ~ X/2 days.

Case Studies

Illustration of performance trends and of the process for a particular wave index scheme

- **SCAM**
  - A research prototype for finding copyright violators.
  - For the experiments provides index to a set of newsgroups to allow authors to search for recent illegal copies of their articles.
- **Web Search Engine (WSE)**
  - Several WSE's index a large set of WebPages for a sliding window of n previous days.
  - For a 'generic' WSE, report results for the case where WSE has to index articles for a sliding window of 35 days.
- **TPC-D**
  - A benchmark from the Transaction Processing council.
  - A large database modeling a decision support environment.
  - For *SUPPKEY (att)* on *LINEITEM (rel)*, build a wave index for a window of the past 100 days. A single query is executed ('Pricing Summary Report').

Experimental Results 1

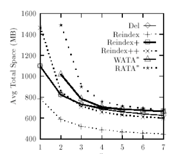


Figure 3: Average space required during SCAM's operation and transition (W = 7).

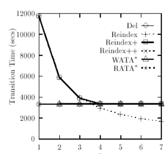


Figure 4: Average transition time for SCAM (W = 7).

- Reports the overall space required (averaged across transitions).
- REINDEX requires the minimal space.
  - Maintains packed indices.
  - Does not have any temporary indices.
- Overall the schemes require less space as n increases
  - Fewer temporary indices required
- Transition time to index new data.
  - Using BuildIndex or AddToIndex ?
  - How many days are reindexed ?
- DEL, WATA, RATA, REINDEX++ execute AddToIndex & incrementally index 1 day.
- REINDEX cost savings of BuildIndex if n>3
- REINDEX+ performs poorly -> executes AddToIndex several times each day

Experimental Results 2

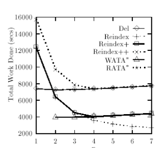


Figure 5: Average work done by SCAM during day (W = 7).

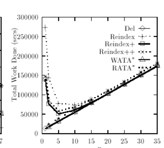


Figure 6: Average work done by WSE during day (W = 35).

- Total work done by different schemes.
- Sensitive to the mix of queries and updates.
  - Best to perform more work at update time in order to get better index(eg packed)
- REINDEX performs well for large n
  - Relative cost of BuildIndex + packed
- DEL, WATA, RATA stable -> incremental additions
- Total work done by WSE with packed shadowing for W = 35.
- REINDEX performs worst
  - higher query volume and window size.
- DEL, WATA and RATA performs best and does the minimal total work –
  - minimal work for reindexing new data + Index probes are cheap for small n.

Experimental Results 3

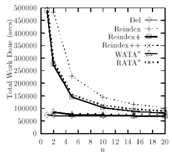


Figure 7: Work done in TPC-D (packed shadowing) during day (W = 100).

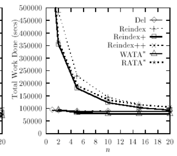


Figure 8: Work done in TPC-D (simple shadowing) during day (W = 100).

- Total work done in TPC-D with packet shadowing.
- REINDEX performs very poorly.
  - Reindexing W/n days each day.
- DEL and WATA perform best.
  - Packet shadowing does deletion while copying -> reducing work

- Total work done in TPC-D with simple shadowing.
- Performs very similar to packed shadowing.
- WATA performs the minimal amount of work -> performs well as n increases
  - Number of expired days stored in the indices decrease as n increases.

Experimental Results 4

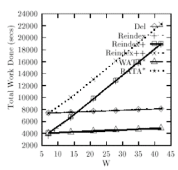


Figure 9: Work done during day by SCAM with W (n = 4).

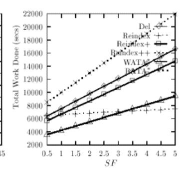


Figure 10: Work done during day by SCAM with SF (W = 14, n = 4).

- How do schemes scale with increasing W ?
- DEL, WATA, RATA scale very well
  - Index a constant number of days every day
- Since reindexing schemes index W/n days each day, work done increases with the size of W.

- How does turnover rate affect the work done ?
- REINDEX scales best
  - does not use incremental indexing
- WATA still scales best for SF < 3

Advantages of Using the schemes

- **Bulk Insert/Delete** : In WATA deletions are performed in bulk by throwing away a whole index. Similarly, it may be efficient to reindex data, like in REINDEX, REINDEX+ and REINDEX++, if the constituent index size is reasonably small.
- **Better Structured Index** : REINDEX may be more costly because it rebuilds indexes from scratch, but this rebuilding can often lead to a better structured index. Such an index could lead to more efficient query processing.
- **Simple Code** : With REINDEX, REINDEX+, REINDEX++, WATA and RATA the scheme do not use complex deletion code. This could be a great advantage if we are implementing our system from scratch.
- **Legacy Systems** : Some information retrieval indexing packages do not implement deletes at all. In such cases DEL can not used .
- **Query Performance** : Each scheme presented has multiple indexes, this creates more work for queries. However when query volume may be relatively low and data volumes may be high, the high query costs may be amortized by the savings under some of the categories listed above.

Conclusions

- One of the first schemes to index data from a past window of days.
- Several techniques proposed for building indices on *temporal data*.
- Different techniques perform differently in different environments (volume of input data, query patterns, index lengths).
- In future, analyze how different indices perform when multiple disks are available.

Discussion Points

- Better techniques to index temporal data ? Or maybe faster operations
  (e.g. Replace(I, d_{new}, d_{old}))?
- How would these schemes scale for frequently updated windows where tuple size is very small ?
- Can they be used/modified to take care of bursty or out-of-order data. (data from sensors on a highway)?
- Do the experiments appropriately demonstrate the performance of the different schemes ?
- Is the scope too broad for one paper ?

Questions ?