UPI: A Primary Index for Uncertain Databases

Presented by
Hideaki Kimura
Brown University

With
Sam Madden
MIT CSAIL

Stan Zdonik
Brown University
Introduction: Uncertain Databases

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Exist?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Brown: 80%, MIT: 20%</td>
<td>90%</td>
</tr>
<tr>
<td>Bob</td>
<td>MIT: 95%, UCB: 5%</td>
<td>100%</td>
</tr>
<tr>
<td>Carol</td>
<td>Brown: 60%, U. Tokyo: 40%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Possible World 1

- Alice: Brown
- Bob: MIT

Possible World 2

- Bob: UCB

Querying over Possible Worlds

\[0.9 \times 0.8 \times 0.95 \times 0.2 = 13\%\]

\[0.1 \times 0.05 \times 0.2 = 0.1\%\]
Ex) DBLP with Uncertain Affiliation

- DBLP: 1.3M Papers and 0.7M Authors
- Complemented Author Affiliation

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>David DeWitt</td>
<td>Wisconsin: 40%, MS: 20%, Columbia: 13%, ...</td>
<td>US: 100%</td>
</tr>
</tbody>
</table>

Zipfian Distribution
Indexes for Uncertain Database

- Special Secondary Indexes
  Ex) PII (Probabilistic Inverted Index)

<table>
<thead>
<tr>
<th>Institution&lt;sup&gt;p&lt;/sup&gt;</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>[Alice], [Carol]</td>
</tr>
<tr>
<td>MIT</td>
<td>[Bob], [Alice]</td>
</tr>
</tbody>
</table>

- But, None Has Explored Primary Index

**Goal**

Build Primary Indexes Over Uncertain Attributes
Why Primary Index Matters?

• Much Faster than Secondary Index

• Benefits Correlated Secondary Indexes
  ▫ Faster Secondary Indexes
  ▫ Smaller Secondary Indexes

For more details:
CORADD
Thursday 2p
At Falcon
Problem: Attribute Uncertainty

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution&lt;sub&gt;p&lt;/sub&gt;</th>
<th>Exist?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Brown: 80%, MIT: 20%</td>
<td>90%</td>
</tr>
<tr>
<td>Bob</td>
<td>MIT: 95%, UCB: 5%</td>
<td>100%</td>
</tr>
<tr>
<td>Carol</td>
<td>Brown: 60%, U. Tokyo: 40%</td>
<td>80%</td>
</tr>
</tbody>
</table>

Cluster on most probable possible value?

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>Alice, Carol</td>
</tr>
<tr>
<td>MIT</td>
<td>Bob</td>
</tr>
</tbody>
</table>

Replicate tuples into inverted index?

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>Alice, Carol, ...</td>
</tr>
<tr>
<td>MIT</td>
<td>Alice, Bob, ...</td>
</tr>
</tbody>
</table>

Too Large for Long-tail distribution (e.g., 100 values with 0.1%)

SELECT WHERE Inst.=MIT

Alice?
Overview

• UPI
  ▫ Cutoff Index
  ▫ Choosing Threshold

• Secondary Index on UPI

• Fractured UPI

• (Continuous UPI)

• Experiments
UPI: Heap and Cutoff Index

### Name | Institution | Exist?
--- | --- | ---
Alice | Brown: 80%, MIT: 20% | 90%
Bob | MIT: 95%, UCB: 5% | 100%
Carol | Brown: 60%, U. Tokyo: 40% | 80%

**Heap: Sorted by (Inst., Prob)**

<table>
<thead>
<tr>
<th>Institution</th>
<th>Tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown (72%)</td>
<td>Alice</td>
</tr>
<tr>
<td>Brown (48%)</td>
<td>Carol</td>
</tr>
<tr>
<td>MIT (95%)</td>
<td>Bob</td>
</tr>
<tr>
<td>MIT (18%)</td>
<td>Alice</td>
</tr>
<tr>
<td>UCB (5%)</td>
<td>Bob</td>
</tr>
<tr>
<td>U. Tokyo (32%)</td>
<td>Carol</td>
</tr>
</tbody>
</table>

**Cutoff Index: Sorted by (Inst., Prob)**

<table>
<thead>
<tr>
<th>Institution</th>
<th>#ID</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCB (5%)</td>
<td>[Bob]</td>
<td>MIT</td>
</tr>
</tbody>
</table>

**Cutoff Entries with Less than C probability (Cutoff Threshold)**
Answering Queries with UPI

- Probabilistic Threshold Query (PTQ)

```
SELECT * FROM Author WHERE Inst.=UCB
With: Probability \geq QT (Query Threshold)
```

<table>
<thead>
<tr>
<th>Institution</th>
<th>Tuple</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT (95%)</td>
<td>Bob</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>UCB (90%)</td>
<td>Dan</td>
</tr>
<tr>
<td>UCB (20%)</td>
<td>Emily</td>
</tr>
</tbody>
</table>

- If $QT \geq C$ (e.g., $QT=20\%$), Sequentially Read
- If $QT < C$ (e.g., $QT=5\%$), follow Cutoff-ed pointers

$C=10\%$
Choosing Cutoff Threshold

Selective Case (#Pointers=300)

- Faster, but Larger
- Slower, but Smaller

More Cutoff-ed Pointers
Analytic Cost Model

- Used to Determine ‘C’
- Based on Value/Probability Histograms

**Histograms (Inst.)**

<table>
<thead>
<tr>
<th>Value</th>
<th>#Keys</th>
<th>Prob.</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Br-Bs</td>
<td>30,000</td>
<td>10%-15%</td>
<td></td>
</tr>
<tr>
<td>Bs-Bt</td>
<td>31,000</td>
<td>15%-25%</td>
<td></td>
</tr>
<tr>
<td>Bt-Bz</td>
<td>30,500</td>
<td>25%-40%</td>
<td></td>
</tr>
</tbody>
</table>

C, Query

= #Pointers * SeekCost?
#Pointers and Query Cost

Non-Selective Case (#Pointers=37000)

Cost Model

\[ \text{Cost}_{\text{cutoff}} = \text{Cost}_{\text{fullscan}} \cdot \text{Selectivity} + f(#\text{Pointers}) \]

\[ f(x) = \text{Cost}_{\text{fullscan}} \left( \frac{1 - e^{-kx}}{1 + e^{-kx}} \right) \]

Logistic function

Query Runtime [sec]

Cutoff Threshold C

Query Runtime [sec] vs. Cutoff Threshold C for different QT values (0.05, 0.15, 0.25). The graph shows the saturation effect as the cutoff threshold increases.
Secondary Index on UPI

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Country</th>
<th>Exist?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Brown: 80%, MIT: 20%</td>
<td>US: 100%</td>
<td>90%</td>
</tr>
<tr>
<td>Bob</td>
<td>MIT: 95%, UCB: 5%</td>
<td>US: 100%</td>
<td>100%</td>
</tr>
<tr>
<td>Carol</td>
<td>Brown: 60%, U. Tokyo: 40%</td>
<td>US: 60%, Japan: 40%</td>
<td>80%</td>
</tr>
</tbody>
</table>

**UPI**

<table>
<thead>
<tr>
<th>UPI</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown (72%)</td>
<td>Alice</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MIT (95%)</td>
<td>Bob</td>
</tr>
<tr>
<td>MIT (18%)</td>
<td>Alice</td>
</tr>
</tbody>
</table>

SELECT ... WHERE Country=US

Secondary Index on (Country)

<table>
<thead>
<tr>
<th>Country</th>
<th>Pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td>US (100%)</td>
<td>[MIT]</td>
</tr>
<tr>
<td>US (90%)</td>
<td>[Brown], [MIT]</td>
</tr>
</tbody>
</table>

Tailored Index Access

- Store Multiple Pointers
- Determine which to use at runtime

Alice is also in MIT...
Maintaining Primary Index

• Random Read/Write
  ▫ Disk Seeks for each INSERT
  ▫ Much Slower than Unclustered Heap

• Fragmentation
  ▫ B+Tree nodes Split/Merge
  ▫ Internal/External Fragmentation
  ▫ Makes SELECTs Slower over time
Fractured UPI

Main Fracture
- UPI Heap File
- Cutoff Index
- 2ndary Index
- 2ndary Index

Fracture 1
- Delete Set

New Fracture
- Delete Set

Query Independently
- SELECT
- INSERT
- DELETE

Dump
- Insert Buffer (On RAM)
Experiments

- Environments
  - C++ & BerkeleyDB 4.7 on Fedora Core 11
  - Quad-Core, 4GB RAM, 10k RPM SATA HDD
- Dataset: DBLP w/ Uncertain Affiliation
- Compared With PII (and U-Tree in paper)
- Cold Start
Query Runtime: PII vs. UPI

Q1: SELECT * FROM Author WHERE Institution=x

UPI Causes Much Fewer Disk Seeks
Cutoff Index Cost Model (1)
Selective Case (Q1, #Pointers=300)

Real Runtime

Estimated Runtime

Query Runtime [sec] vs. Cutoff Threshold C

- QT=0.05
- QT=0.15
- QT=0.25

Cutoff Threshold C vs. Query Runtime [sec]
Cutoff Index Cost Model (2)

Non-Selective Case (Q1, #Pointers=37000)

Real Runtime

Estimated Runtime

Query Runtime [sec]

Cutoff Threshold C

Cutoff Threshold C

QT=0.05  QT=0.15  QT=0.25

0 0 0

0.1 0.1 0.1

0.2 0.2 0.2

0.3 0.3 0.3

0.4 0.4 0.4

0.5 0.5 0.5
Tailored Secondary Index Access

Q2: SELECT Journal, COUNT(*) FROM Publication WHERE Country=x GROUP BY Journal

<table>
<thead>
<tr>
<th></th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read PII</td>
<td>110 [ms]</td>
</tr>
<tr>
<td>Read UPI</td>
<td>3,200 [ms]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read PII</td>
<td>110 [ms]</td>
</tr>
<tr>
<td>Tailor</td>
<td>33 [ms]</td>
</tr>
<tr>
<td>Read UPI</td>
<td>500 [ms]</td>
</tr>
</tbody>
</table>

Query Runtime [sec]

- PII on unclustered heap
- PII on UPI
- PII on UPI w/ Tailored Access
Fractured UPI

<table>
<thead>
<tr>
<th></th>
<th>Insert 10%</th>
<th>Delete 1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclustered Heap</td>
<td>8 sec</td>
<td>75 sec</td>
</tr>
<tr>
<td>UPI</td>
<td>650 sec</td>
<td>212 sec</td>
</tr>
<tr>
<td>Fractured UPI</td>
<td>4 sec</td>
<td>0.03 sec</td>
</tr>
</tbody>
</table>

Query Runtime [sec] vs Processed Insert Batches

- Unclustered heap
- UPI
- Fractured UPI

Fragmentation
More Fractures
Conclusion

- **Primary Index for Uncertain DB**
  - Essential for Non-Selective Queries
  - Benefits Correlated Indexes
- **UPI**
  - Cutoff Index and Cutoff Threshold
  - Tailored Secondary Index Access
  - Continuous UPI
  - Fractured UPI
Future Work

• Applying to other types of queries
  ▫ Top-k Query: UPI as Tuple Access Layer

• Database Designer for Uncertain DB
  ▫ Index/MV Recommendation
  ▫ Correlation Analysis in Uncertain DB
  ▫ Incremental Design