Database-related research at ITU

Rasmus Pagh
IT University of Copenhagen

Database Tuning, Spring 2009
Today

- ITU research in databases:
  - Hashing on external memory: - Can it be improved? - Without a hash function?
  - Scalable computation of acyclic joins.
  - Algorithms for combinations of relational algebra operations: - Cyclic joins, - Join+project
  - Generalization of bitmap indexes and B-trees.
  - Better association mining.

- End notes on exam, etc.

- **Cancelled**: Rasmus Resen Amossen on H-Store.
A better external hash table?

Ideas (some old, some new):

- Keep hash table blocks very close to full. (There will be many overflows.)
- Keep in each block a buffer of updates that need to be done in the overflow chain - almost all updates use 2 I/Os.
- Maintain records ordered such that almost all searches only need 1 I/O.

Work with Morten Skaarup Jensen
Algorithmica 2008
Hashing with no hash function?

Work with Mette Berger, Esben Rune Hansen, Mihai Pătrașcu, Milan Ruzic, and Peter Tiedemann, SPAA 2006

- Problem with hashing: May fail to have good performance. Would like deterministic data structure with same performance.

- Most experts believe this is not possible on a "normal computer", using little space.

- But what if we have more resources - e.g., an array of 64 disks that we can access in parallel? (Such storage systems exist.)
Deterministic load balancing

Idea:

• Associate (in a clever way) with each possible key \( x \) a **fixed** set of disk blocks where it may be stored.

• One possible block on each disk - lookup in 1 parallel I/O.

• When inserting a key \( x \), check all possible positions (in parallel) and insert \( x \) in the *least full* block.

• This **always** works if (roughly):
  
  • Blocks have room for \( \log N \) keys.
  
  • There are \( \log(\#\text{possible keys}) \) disks.
**Acyclic join**

- **Definition.** A join of relations $R_1,\ldots,R_k$ is *acyclic* if there exists a tree with vertices $R_1,\ldots,R_k$ such that the join can be expressed as a `SELECT` `FROM` $R_1,\ldots,R_k$ `WHERE` ... with equality conditions only between adjacent relations.

- **Example:**

```
R_1(a,b)
R_2(a,c)
R_3(a,d)
R_4(y,z)
R_5(z,b)
```

This is a cyclic join.
Special case: Star schema

- **Star schema**: The attribute sets of the relations are disjoint, except $R_1$ which contains an attribute from each of $R_2, ..., R_k$.

- **Example**:

  $R_1(a, b, c)$
  $R_2(a, x)$
  $R_3(b, y)$
  $R_4(c, z)$
Star schema join

- Natural join of $R_1, \ldots, R_k$ having a star schema.
- Any join not involving $R_1$ is a cartesian product.
- Best query plan usually highly unbalanced:

  All intermediate results, and the final result, may have roughly the same size as $R_1$. 

$$\text{height } \Omega(k)$$
Worst-case complexity

- Let $n$ denote the size of input ($\approx$ size of output).
- Total size of inputs to binary joins $\Omega(kn) \Rightarrow$
  \[
  \begin{cases}
  \Omega(kn) & \text{time on a RAM} \\
  \Omega(kn/B) & \text{I/Os in the I/O model}
  \end{cases}
  \]
- Pipelining and indexing don’t help asymptotically.
- Next: Eliminating the dependence on $k$. 
New approach

- Abandon the idea of intermediate results.
- Compute relationship between input and output.
New approach

- *Abandon* the idea of intermediate results.
- Compute relationship between input and output.

**R₁**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>c</td>
</tr>
</tbody>
</table>

**R₂**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>a</td>
<td>α</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>a</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>b</td>
<td>β</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>b</td>
<td>β</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>b</td>
<td>β</td>
<td>Z</td>
</tr>
</tbody>
</table>

**R₃**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>O</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>T</td>
</tr>
</tbody>
</table>

Sort by row numbers: 1, 2, 3, 4, 5
Implementation

1. Count number of tuples matching each tuple in $R_1$. 

<table>
<thead>
<tr>
<th>R₁</th>
<th>R₂</th>
<th>R₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>2×1</td>
<td>a</td>
<td>0×2</td>
</tr>
<tr>
<td>1×1</td>
<td>b</td>
<td>0×1</td>
</tr>
<tr>
<td>1×2</td>
<td>b</td>
<td>1×2</td>
</tr>
<tr>
<td>0×2</td>
<td>c</td>
<td>2×1</td>
</tr>
</tbody>
</table>
Implementation

1. Count number of tuples matching each tuple in $R_1$.

2. Enumerate the output tuples.
General result

- Acyclic join of $k$ relations.
- Let $n$ denote the input size plus the size of data involved in the equality conditions of the join. Let $z$ denote the output size.
- We can compute the join in
  - $O(\text{sort}(n+z))$ I/Os on external memory
  - $O(n+z)$ expected time on a RAM
Better query evaluation

Work with Rasmus Resen Amossen, ICDT 2009, and Anna Pagh, PODS 2006

– Consider SQL statements such as:

1. SELECT DISTINCT a,c FROM R1, R2
   WHERE R1.b=R2.b;
   -- e.g. find all co-stars in movies

2. SELECT * FROM R1,R2,R3
   WHERE R1.b=R2.a AND R2.b=R3.a AND R3.b=R1.a
   -- e.g. find all friends who know each other

– It turns out that traditional DBMSs (that evaluate queries using a sequence of relational algebra operations), can be outperformed, at least in worst case. Practical impact?
Query 1: Join + project

Rows with A- and B-values that occur “many times”

Rows with remaining A- and B-values

“Block-nested loop” join

Sort-merge join

Rows with B- and C-values that occur “many times”

Rows with remaining B- and C-values
Query 2: Cyclic join (3-cycle)

Analysis: Each tuple in $R_3$ is a possible match $O(n/M)$ times.

$O(n^2/(MB) + \text{sort}(n))$ I/Os

(previously: $O(n^2/B)$ I/Os)
Secondary indexing in 1D

Work with S. Srinivasa Rao, PODS 2009

- Find the set of tuples satisfying a range condition (e.g., $10 \leq \text{age} \leq 20$).
- Traditionally supported by search trees ($B$-trees): Results represented as reference lists.
- Bitmap indexes faster in some cases: Results represented as (compressed) bitmaps.
Range queries

- To answer a range query, one can compute the bitwise OR of the corresponding bitmaps (compressed).
- This is not always I/O optimal:
  - Assume k values, each with n/k occurrences
  - Compressed bitmaps use $O\left(\frac{n}{k}\log k\right)$ bits each
  - Range query on k/2 values needs to read $O(n \log k)$ bits, but the result can be represented in n bits.
Bitmap index performance

density of output

size of range

optimal performance

log factor from optimal
Bucketing

- **Idea**: Precompute compressed bitmaps for certain ranges, called “buckets”.

- May continue recursively, “multi-resolution” bucketing (Sinha and Winslett, 2007).

```plaintext
22 25 28 30 33 34 39 42
```
Multi-resolution bitmap index
Sinha and Winslett, 2007

density of output

size of range

time/space tradeoff

optimal performance

log factor from optimal
Space-time trade-off?

- Seemingly, there is a trade-off: Using more levels of resolution gives faster queries, but requires more space.

- Our main observation: By choosing the levels of resolution in the right way, one can get optimal space and query time (in the I/O model).
Optimal 1D secondary index

Pagh and Rao, PODS 2009

density of output

size of range

linear space

optimal performance

log factor from optimal
Proven practical impact

- Bayer & McCreight 1972
- Sinha & Winslett 2007
- O’Neil & Quass 1997
- Chan & Ioannidis 1989
- Pagh & Rao 2009

Compressed bitmap indexing
All resolution levels

Complete binary tree

All 1s bitmap

Bitmaps at level $j$ have $n/2^j$ elements, and use $j$ bits of space per element.

Individual bitmaps
Selected resolution levels

- 2 bits/element
- 4 bits/element
- 8 bits/element
- 16 bits/element
- log(k) bits/element
Queries

- 2 bits/element
- 4 bits/element – “lower bound” on output size
- 8 bits/element – most of the data is read from here
- 16 bits/element
- \( \log(k) \) bits/element
Association mining algorithms

Agrawal & Srikant ’94, [PCY ’95], [SON ’95], Brin et al. ‘97

Data → Column sampling → Frequent pairs → Extract → Related rows

Data → Sketching → Biased pair sampling

Campagna & Pagh, 2009
Our results

- Avoids frequent pairs bottleneck — **worst case** performance reduced from quadratic to near-linear.
- **Price paid**: False negative and false positive probability.
- **Experiments**:
  - Computation time typically 1 order of magnitude lower than competing methods.
  - On-going work on evaluating space usage.
Thesis/project topics

- Many possibilities. Starting point can be your own idea (usually needs “sharpening” to work well), or a project proposed by me.

- Some of the things I would find most interesting are:
  - Trying out the new 1D indexing methods.
  - Semi-supervised association rules.
  - Practical variant of join-project procedure?
  - Using new hardware (graphics cards, flash devices,...) to implement data management tools.
Exams

- Exams are on June 15 and 16.
  - Schedule out soon after project hand-in.
  - Oral without preparation; individual.
  - No ”presentation”. We will ask questions, taking the project report as a starting point.
  - Main focus on **skills**, not knowledge (course goals).
  - But of course, tuning skills often require knowledge.
- Q&A session before exam: June 5?, or June 13?
Recommendations from “aftagerpanelet”

February 2009

- The ability to “Learn how to learn” was considered an essential skill by all panel members.
- It is important not just to teach a certain curriculum, but encourage the students to be curious and explorative.
- Representatives ... in the panel have emphasized analytical and problem solving skills as highly sought.
- Communicative competences are also in demand ... it is important to have strong technical communicative skills.
Endnote: Pioneers of failed flight