Evaluating Queries

Query Processing

Query Processing: Overview

physical query? (have operators: ROWACCESS, FILESCAN, INDEXSCAN)

Query Processing: Example

physical query? (have operators: ROWACCESS, FILESCAN, INDEXSCAN)
Classical Example: Sorting

- why?
  - ORDER BY
  - duplicate removal (intersection, union, DISTINCT)
  - sort/merge for join
- how?
  - in main memory easy: quick sort
  - issue: large relations don’t fit in main memory

Scheduling

- scheduling of access is important
- Example:
  - 3 pages of main memory,
  - least recently used replacement policy
  - pages in main memory:
    - page 1: 1, 5, 9, .., 37
    - page 2: 2, 6, 10, .., 38
    - page 3: 3, 7, 11, .., 39
    - page 4: 4, 8, 12, .., 40
  - how many page accesses to read 1-40?

Another Query Plan Example

```sql
select e.name, s.name
from employee as e, employee as s
where e.superid = s.id
```

QEP!
External Sorting I

- goal: minimize page transfers
- assumptions:
  - data stored in n pages
  - m << n pages fit into main memory
- solution: sort in runs (temporary sorted subfiles that get merged on disk)

External Sorting: Algorithm

- partition input file into blocks of m pages
- sort internally, write-out into n/m initial “runs”
- at each level of the recursion
  - merge m-1 runs into a new run
  - use 1 page in main memory for each run
  - 1 page in main memory to create merged page
  - write output page if full/reload input page when processed

External Sorting: Performance

- analyze set-up phase
- how many page I/Os at each recursive level?
- how many recursive levels?
- overall analysis?
- internal sort?
Rectangle Intersection, Again

- original solution: sweepline
  - event list: left/right x-coordinates of rectangles
  - active list: rectangles at current x-value

- needed for spatial join (on overlap)
- external technique: distribution sweeping

Orthogonal Line Segments

input: S set of vertical/horizontal line segments
output: pairs of intersecting segments

sweepline algorithm:
- events: x-min coordinates
- active list: horizontal segments at x-value
- when vertical segment is encountered: range query
- works in time O(n log n + k)

Orthogonal line segments (EM)

- assumption
  - n pages of data
  - m buffer pages in main memory
- external sort (x-min): O(n log m n)
- split into m horizontal strips of n/m horizontal segments each
- one active list (stored externally) for each slab (can be read in parallel with others one block at a time, since we have m pages in main memory)
Process center pieces

- why can’t we process end-pieces the same way?
- what to do about end-pieces?

End-pieces

- apply method recursively
- analysis
  - initial set-up
  - each recursive level
  - depth of recursion?
- overall running time analysis

Rectangle Intersection

- input: B, R sets of rectangles
- output: all \((b, r)\) in \(B \times R\) with \(b\) intersecting \(r\)

- solution:
  - similar to line segment problem, separate lists for red and blue rectangles
  - but: rectangles don’t fit into strips, they can span multiple strips
  - problem: intersection between \((b, r)\) might be reported multiple times
  - so naïve adaptation gives factor of \(m\)
Rectangle Intersection

Solution:
- separate list for each interval of strips and each color: \( L_{b,k} \), \( L'_{b,k} \)

Analysis?

Spatial Join
- join on: topology (overlap, disjoint, contain, ...), geometry (distance, direction)
- consider overlap only
  - filter: overlap of mbbs
  - refinement: overlap of geometries
- depends on indexes available
Spatial Join Algorithms

- no indexes
  - distribution sweep
  - hash-join algorithm
- single index
  - INL (indexed nested loop)
- two R-tree indexes
  - synchronized tree traversal
- two linear trees

single index: INL

for each o in non-indexed relation
  perform range query with o.mbb on indexed relation

no index

hash-join algorithm
  - assume buckets fit into main memory
  - hash keys of relations R, S into buckets
  - load smaller bucket, compare to corresponding bucket

Example: R: 2000 records, S: 500 records
  hash into 100 buckets
  page I/O? (read R, S, write buckets, join)
no index for spatial data

- hash-join depends on join condition being equality: overlapping rectangles won’t hash to same bucket
- solution:
  - buckets determined by rectangles
    - may overlap (no redundancy) or be disjoint (redundancy)

hash-join (overlapping)

1. partition R
   - all buckets roughly same size
   - buckets should fit into main memory
   - minimal overlapping
2. assign rectangles of S to buckets of R
   - S rectangles might be duplicated
3. join buckets (load smaller bucket into main memory, scan other bucket)

joining two linear trees

- raster trees: traditional join
- general linear tree (e.g. linear quadtree or z-ordering tree)

Property:
\[ C_z \text{ is contained in } C_{z'} \text{ if and only if } z' \text{ is prefix of } z \]

- can use to test overlap
**joining two z-ordering trees**

- replace each entry \((z, \text{oid})\) with intervals \((z, \text{ssc}(C_z))\), where \(\text{ssc}(C_z)\) is lower-right corner of \(C_z\)
- two squares overlap iff their intervals overlap
- store each list in a stack

```
A
B
C
D
E
F
G
H
I
J
K
```

**R-trees**

- naïve recursion
- restricted recursion
- sweep-line

**R-trees recursively**

- STT (Synchronized Tree Traversal)

```plaintext
STT (Node N_i, Node N_j): set of pairs of ids

begin
result set of pairs of ids, initially empty
for all \(x, y\) in \(N_i\) do
  if \(x < y\) then
    result := result \cup \{(x, y)\}
  else
    \(N_i' := \text{ReadPair}(x, y); N_j' := \text{ReadPair}(x, y);\)
    \(\text{result} := \text{STT}(N_i', N_j');\)
  end if
end for
return result
end
```

- I/O performance ok
- CPU cost high
R-trees recursively, improved

- STT(R1, R2)
- do we need to look at every combination of (R3, R4, R5) and (R6, R7)?

R-trees, sweepline

- why not use red/blue intersection algorithm we saw earlier?
- Asymptotics vs constants
  - greedy approach:
    - order red/blue sets
    - keep picking leftmost rectangle r
    - keep testing rectangles s of opposite color so that s.xmin < r.xmax
    - remove leftmost rectangle

Example

- analysis (bad case?)
Building Query Execution Plans

select intersect(l.shape, c.shape)
from county c, land_use l
where c.county_name = 'San Jose'
and overlaps(l.shape, c.shape);

pipelined execution possible:

Iterators

pipelined execution possible
- iterators (open, next, close)
- e.g. rowaccess, retrieve one record at a time
issues:
  - refinement not included yet
  - CPU time plays a role

Example

- spatial join between roads and land-use
- both relations have R-tree

Problems:
- random access to records (high I/O)
- repeated access to same records
- so refinement needs to be done carefully
Sequencing

- Assume 4 pages fit into main memory; look at schedule (a)

Sequencing: Segment Sort

- k: number of pairs (Lx, Ridy) that fit into m-1 pages
- Load k pairs (LIDx, Ridy) into m-1 pages
- Sort on LID, access land-use replace LID with L records
- Sort on Rid load records from Road using mth page, perform refinement step for each record

Why Not

- Sort (LID, RID) after STT?
- Means we can't pipeline: sorting is a blocking operator
Multiway Joins

Sources

- Vassilakopoulos, Papadopoulos, Spatial databases, IGI, 2005