CS 347 review session

Selected topics from CS 145 and CS 245
Diagrams and notes taken from CS 245

14th April 2016
CS 347 review session

- Index Structures
- Query Optimisation
- Failure Recovery
- Concurrency Control
Index Structures
An Index

An indirect shortcut derived from pointing into a greater volume of values / data

Example: index in a book
Hashing

Dynamic hashing:

- Linear hashing
- Extensible hashing
key $\rightarrow h(\text{key})$
Utilisation

Utilisation = # keys used

Total # keys that fit

- Key metric, should be between 50-80%
Extensible hashing

Use $i$ of $b$ bits of the hashed key $h(key)$
Extensible hashing

Pros

● Can handle growing files
  ○ Less wasted space
  ○ With no full reorganisations

Cons:

● Indirection
  ○ Okay if it fits in memory
● Directory grows by doubling in size
  ○ May no longer fit in cache / memory
Linear hashing

Use $i$ low order bits of the hashed key $h(key)$

Grows linearly

$m = \text{max used block #}$

Rule:

- if $h(key)[i] \leq m$
  - then bucket $h(key)[i]$
- else bucket $h(key)[i] - 2^{i-1}$

Grows when utilisation breaks threshold
b=4 bits, i=2, 2 keys/bucket

What happens if you try to insert 0101?
Linear hashing

Pros

● Can handle growing files
  ○ Less wasted space
  ○ With no full reorganisations
● No indirection like extensible hashing

Cons:

● Can still have overflow chains
Query Optimisation
Query optimisation

Idea: you want to execute your query in the fastest manner with minimum resources

- **Normalise**
  - Convert input into a normal form in Relational Algebra

- **Logical optimisation**
  - Find an equivalent plan that is more efficient
    - Minimise # of tuples at each step

- **Physical optimisation**
  - Find an algorithm with lowest IO cost
Relational Algebra

Some operators:

- Selection $\sigma$
- Projection $\pi$
- Union $U$
- Natural join $\bowtie$
- Cartesian product $\times$
Logical optimisation

General idea: reduce the number and size of tuples at each stage by pushing projections and selections down

Examples:

```
SELECT B, D
FROM R(A,B,C), S(C,D,E)
WHERE R.A = "c"
AND S.E = 2
AND R.C = S.C
```
Cost Estimation

● (logical) Size of results
  ○ Number of tuples
  ○ Size of each tuple, in bytes

● (physical) # of IOs
Estimating size of results

The DB keeps statistics to help you:

- T(R): # tuples in relation R
- S(R): # bytes of a tuple in R
- B(R): # blocks to hold all tuples in R
- V(R,A): # distinct values of attribute A in R

Rules for operations:

- Cartesian product: \( T(W) = T(R) \times T(S) \)
- Selection: \( T(W) = T(R) / V(R,A) \)
- Projection: \( T(W) = T(R) \)
- Natural join: \( T(W) = T(R) \times T(S) / \max(V(R,A), V(S,A)) \)
Estimating # IOs

Cost of physical plan depends on factors like:

- Join algorithms
  - Iteration, merge, index, hash
- Memory management
  - Tuples of relation stored physically together / contiguous
- Sorted on join attribute?
- Indexes exist?
- Parallel processing
Hash join of $R \bowtie S$

Key idea: reduce ‘search’ space by considering only tuples from $R$ and $S$ that hash to the same bucket.

1. Read $R$, hash into buckets and write buckets out
2. Repeat for $S$
3. Read one bucket of $R$ and build an in-memory hash table
4. Read corresponding bucket of $S$ and hash probe

Assumption: $R$, $S$ not sorted.
Optimisation: keep one bucket of $R$ from step 1 in memory during step 2