CS 143  Introduction to Database Systems

TA: Song Jiang
05/15/2018
Join Operations: overview

• Types of Join operations
  • Nested-loop join
  • Index Join
  • Sort Merge Join
  • Hash Join

• Cost Analysis

• Sample Question
Cost Model

• Total number of disk blocks that have been read/written
  • Data are load from disk to memory in the unit of block
  • Count only when a disk block is being loaded into memory
  • Ignore the time of in-memory processing
Running Example Setup

• Two tables R, S
• R has 1,000 tuples; S has 10,000 tuples
• 10 tuples/block ($b_R=100$ blocks, $b_S=1,000$ blocks)
• Memory buffer: $M=22$ blocks
Nested-Loop Join

- Read a block from R at a time
  - For each tuple in R, compare it with each tuple in S
- Cost: $b_R + |R|b_S = 100 + 1000 \times 1000 = 1000100$
Nested-Loop Join

• What if we read S first?
• Cost: $b_S + |S|b_R = 1000 + 10000 \times 100 = 1001000$
• Worse results!
Block Nested-Loop Join

- Read a block from R at a time
  - For each block in R, compare it with each block in S
- Cost: $b_S + b_R \times b_S = 1000 + 100 \times 1000 = 101000$
Block Nested-Loop Join

• What if we read R first?
• Cost: $b_R + b_S \times b_R = 100 + 100 \times 1000 = 100100 < 101000$
• Summary: Use the smaller table on the left (or outer loop)
Sort-Merge Join

• Two stage algorithm
  • Sort stage: Sort R and S
  • Merge stage: Merge sorted R and S
Sort-Merge Join

- Read R and S blocks one block at a time
  - Merge...
- Cost: $b_R + b_S = 1000 + 100 = 1100$
Hash Join

- Hash function $h(v)$, range $1 \rightarrow k$

Algorithm
1. Hashing stage (bucketizing): hash tuples into buckets
   - Hash $R$ tuples into $G_1, \ldots, G_k$ buckets
   - Hash $S$ tuples into $H_1, \ldots, H_k$ buckets
2. Join stage: join tuples in matching buckets
   - For $i = 1$ to $k$ do
     - match tuples in $G_i$, $H_i$ buckets
Hash Join – hashing stage

Buckets number:
= M-1
= 22-1 = 21

Bucket size:
= \lceil b_R / (M-1) \rceil
= \lceil 100 / 21 \rceil = 5

Block read: b_R = 100
Block write: \sim b_R = 100

Total: \sim 2b_R = 200
Hash Join – hashing stage

Buckets number:
\[= M - 1\]
\[= 22 - 1 = 21\]

Bucket size:
\[\approx \frac{b_s}{(M - 1)}\]
\[\approx \frac{1000}{21} = 48\]

Block read: \(b_s = 1000\)
Block write: \(\sim b_s = 1000\)

Total: \(\sim 2b_s = 2000\)
Hash Join – join stage

Total cost of two stages: 200+2000+1100 =3300
Index Join

• What if we have an index built on the join key of S?
• Index join cost:
  • I/O for R scanning : $b_R$
  • I/O for index look up: C
  • I/O for tuple read from S: J

General cost: $b_R + |R| \cdot (C + J)$
  
  – $C$ average index look up cost
  – $J$ matching tuples in $S$ for every $R$ tuple
  – $|R|$ tuples in $R$
Index Join

Example 1
15 blocks for index (1 root, 14 leaf), read index into memory first
On average, 1 matching S tuples per an R tuple

How many IO?
index: 15
C: B+ tree: 0 15+b_R+ | R | *(0+1)
J: S tuple: 1 15+100+1000*(0+1)=1115
Index Join

Example 1
40 blocks for index (1 root, 39 leaf), read partial index into memory first
On average, 10 matching S tuples per an R tuple

How many IO?
M-3: partial index (1 root, 18 leaf): 19
C: B+ tree: \(0 \times (18/39) + 1 \times (21/39) = 0.5\)
J: S tuple: 10
\(\text{(M-3)} + b_R + |R| \times (C+J)\)
\(19 + 100 + 1000 \times (0.5 + 10) = 10619\)
Index Join Summary

read partial index into memory first

How many IO?

\[ b^+_R |R| \times (C+J) \]

M-3: partial index (negligible)
C: average index look up cost
J: matching tuples in S for every R tuple
| R | : tuples in R