Database Management and Performance Tuning

Index Tuning

Pei Li

University of Zurich
Institute of Informatics

Unit 6

Acknowledgements: The slides are provided by Nikolaus Augsten and adapted from "Database Tuning" by Dennis Shasha and Philippe Bonnet.

Outline

Outline

Join Strategies – Running Example

- Relations: R and S
  - disk block size: $4kB$
  - R: $n_r = 5000$ records, $b_r = 100$ disk blocks, $0.4MB$
  - S: $n_s = 10000$ records, $b_s = 400$ disk blocks, $1.6MB$
- Running Example: $R \bowtie S$
  - R is called the outer relation
  - S is called the inner relation

Example from *Silberschatz, Korth, Sudarashan. Database System Concepts. McGraw-Hill.*

Join Strategies – Naive Nested Loop

- Naive nested loop join
  - take each record of R (outer relation) and search through all records of S (inner relation) for matches
  - for each record of R, S is scanned
- Example: Naive nested loop join
  - worst case: buffer can hold only one block of each relation
  - R is scanned once, S is scanned n_r times
  - in total $n_r b_s + b_r = 2,000,100$ blocks must be read $(= 8GB)$!
  - note: worst case different if S is outer relation
  - best case: both relations fit into main memory
  - $b_s + b_r = 500$ block reads

Join Strategies – Block Nested Loop

- Block nested loop join
  - compare all rows of each block of R to all records in S
  - for each block of R, S is scanned
- Example: (continued)
  - worst case: buffer can hold only one block of each relation
  - R is scanned once, S is scanned b_r times
  - in total $b_r b_s + b_r = 40,100$ blocks must be read $(= 160MB)$
  - best case: $b_s + b_r = 500$ block reads

Join Strategies – Indexed Nested Loop

- Indexed nested loop join
  - take each row of R and look up matches in S using index
  - runtime is $O(|R| \times \log |S|)$ (vs. $O(|R| \times |S|)$ of naive nested loop)
  - efficient if index covers join (no data access in S)
  - efficient if R has less records than S has pages: not all pages of S must be read (e.g., foreign key join from small to large table)
- Example: (continued)
  - B^+-tree index on S has 4 layers, thus max. $c = 5$ disk accesses per record of S
  - in total $b_r + n_r c = 25,100$ blocks must be read ($= 100MB$)

Join Strategies – Merge Join

- Merge join (two clustered indexes)
  - scan R and S in sorted order and merge
  - each block of R and S is read once
- No index on R and/or S
  - if no index: sort and store relation with $b(2\lceil log_{M-1}(b/M)\rceil + 1) + b$ block transfers (M: free memory blocks)
  - if non-clustered index present: index scan possible
- Example: (continued)
  - best case: clustered indexes on R and S ($M = 2$ enough)
  - $b_r + b_s = 500$ blocks must be read ($2MB$)
  - worst case: no indexes, only $M = 3$ memory blocks
  - sort and store R (1400 blocks) and S (7200 blocks) first: join with 9100 ($36MB$) block transfers in total
  - case $M = 25$ memory blocks: 2500 block transfers ($10MB$)

Join Strategies – Hash Join

- Hash join (equality, no index):
  - hash both tables into buckets using the same hash function
  - join pairs of corresponding buckets in main memory
  - R is called probe input, S is called build input
- Joining buckets in main memory:
  - build hash index on one bucket from S (with new hash function)
  - probe hash index with all tuples in corresponding bucket of R
  - build bucket must fit main memory, probe bucket needs not
- Example: (continued)
  - assume that each probe bucket fits in main memory
  - R and S are scanned to compute buckets, buckets are written to disk, then buckets are read pairwise
  - in total $3(b_r + b_s) = 1500$ blocks are read/written ($6MB$)
  - default in SQLServer and DB2 UDB when no index present

Distinct Values and Join Selectivity

- Join selectivity:
  - number of retrieved pairs divided by cardinality of cross product ($|R \bowtie S|/|R \times S|$)
  - selectivity is low if join result is small
- Distinct values refer to join attributes of one table
- Performance decreases with number of distinct join values
  - few distinct values in both tables usually means many matching records
  - many matching records: join result is large, join slow
  - hash join: large buckets (build bucket does not fit main memory)
  - index join: matching records on multiple disk pages
  - merge join: matching records do not fit in memory at the same time

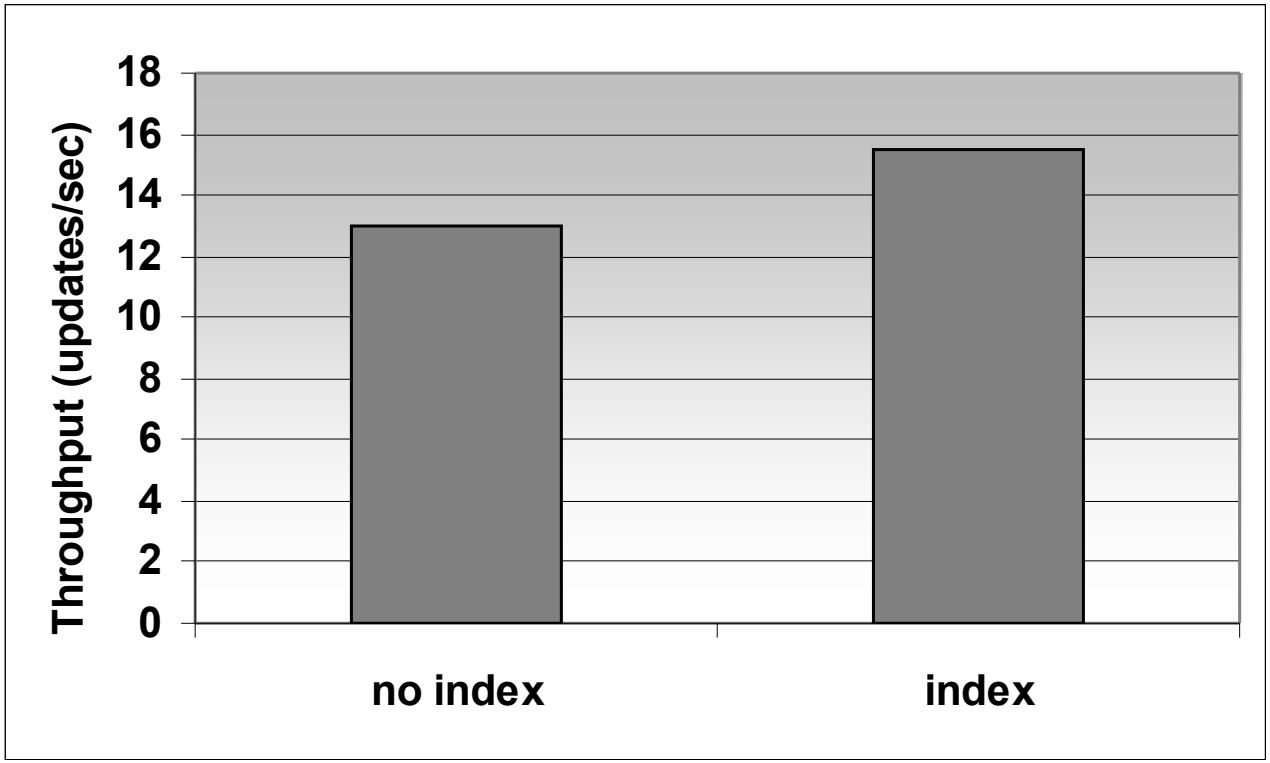
Foreign Keys

- Foreign key: attribute $R.A$ stores key of other table, $S.B$
- Foreign key constraints: $R.A$ must be subset of $S.B$
  - insert in R checks whether foreign key exists in S
  - deletion in S checks whether there is a record with that key in R
- Index makes checking foreign key constraints efficient:
  - index on $R.A$ speeds up deletion from S
  - index on $S.B$ speeds up insertion into R
  - some systems may create index on $R.A$ and/or $S.B$ by default
- Foreign key join:
  - each record of one table matches at most one record of the other table
  - most frequent join in practice
  - both hash and index nested loop join work well

Indexes on Small Tables

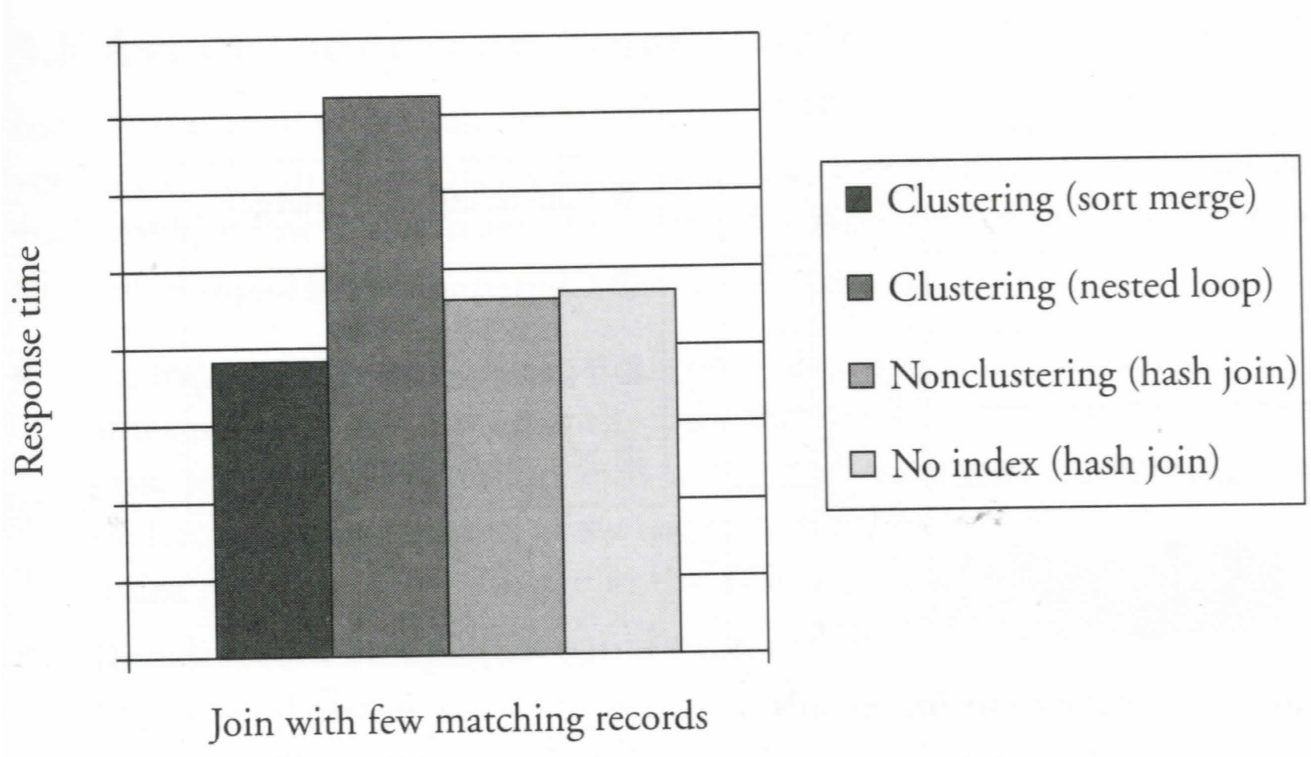
- Read query on small records:
  - tables may fit on a single track on disk
  - read query requires only one seek
  - index not useful: seeks at least one index page and one table page
- Table with large records (∼page size):
  - each record occupies a whole page
  - for example, 200 records occupy 200 pages
  - index useful for point queries (read 3 pages vs. 200)
- Many inserts and deletions:
  - index must be reorganized (locking!)
  - lock conflicts near root since index is small
- Update of single records:
  - without index table must be scanned
  - scanned records are locked
  - scan (and thus lock contention) can be avoided with index

Update Queries on a Small Tables



- Index avoids tables scan and thus lock contention.

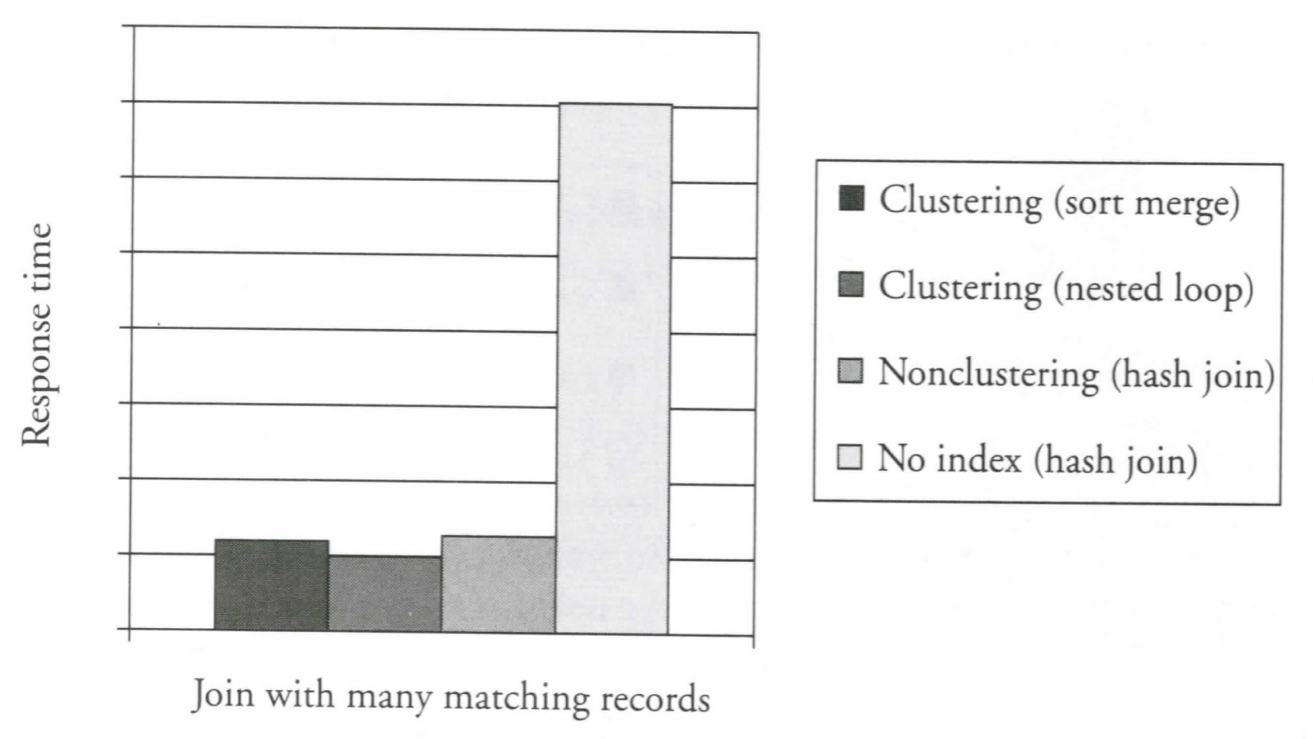
Experiment – Join with Few Matching Records



- non-clustered index is ignored, hash join used instead

SQL Server 7 on Windows 2000

Experiment – Join with Many Matching Records



- all joins slow since output size is large
- hash join (no index) slow because buckets are very large

SQL Server 7 on Windows 2000