Parallel DBMS

Chapter 21, Part A

Slides by Joe Hellerstein, UCB, with some material from Jim Gray, Microsoft Research. See also:
http://www.research.microsoft.com/research/BARC/Gray/PDB95.ppt

Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

1 Terabyte

1.000 x parallel
1.5 minute to scan.

1 Terabyte

Parallelism: divide a big problem into many smaller ones to be solved in parallel.

DBMS: The Success Story

❖ DBMSs are the most (only?) successful application of parallelism.
   - Teradata, Tandem vs. Thinking Machines, KSR.
   - Every major DBMS vendor has some || server
   - Workstation manufacturers now depend on || DB server sales.

❖ Reasons for success:
   - Bulk-processing (= partition ||-ism).
   - Natural pipelining.
   - Inexpensive hardware can do the trick!
   - Users/app-programmers don’t need to think in ||

Parallel DBMS: Intro

❖ Parallelism is natural to DBMS processing
   - Pipeline parallelism: many machines each doing one step in a multi-step process.
   - Partition parallelism: many machines doing the same thing to different pieces of data.
   - Both are natural in DBMS!

Pipeline
Partition

outputs split N ways, inputs merge M ways

Speed-Up
❖ More resources means proportionally less time for given amount of data.

Scale-Up
❖ If resources increased in proportion to increase in data size, time is constant.

Some Terminology

Ideal

degree of ||-ism

Ideal

degree of ||-ism

Architecture Issue: Shared What?

Shared Memory (SMP)       Shared Disk       Shared Nothing (network)
(CLIENTS)                (MEMORY)            (CLIENTS)
(CLIENTS)                (CLIENTS)            (CLIENTS)

Easy to program
Expensive to build
Difficult to scaleup
Sequent, SGI, Sun

Hard to program
Cheap to build
Easy to scaleup
Tandem, Teradata, SP2

What Systems Work This Way
(as of 9/1995)

Shared Nothing
- Teradata: 400 nodes
- Tandem: 110 nodes
- IBM / SP2 / DB2: 128 nodes
- ATT & Sybase: 48 nodes

Shared Disk
- Oracle: 170 nodes
- DEC Rdb: 24 nodes

Shared Memory
- Informix: 9 nodes
- RedBrick: ? nodes

Different Types of DBMS

❖ Intra-operator parallelism
  - get all machines working to compute a given operation (scan, sort, join)
❖ Inter-operator parallelism
  - each operator may run concurrently on a different site (exploits pipelining)
❖ Inter-query parallelism
  - different queries run on different sites
❖ We’ll focus on intra-operator parallelism

Automatic Data Partitioning

Partitioning a table:
- Range
- Hash
- Round Robin

Good for equijoins, range queries
Group-by
- Shared disk and memory less sensitive to partitioning,
  Shared nothing benefits from “good” partitioning

Parallel Scans

❖ Scan in parallel, and merge.
❖ Selection may not require all sites for range or hash partitioning.
❖ Indexes can be built at each partition.
❖ Question: How do indexes differ in the different schemes?
  - Think about both lookups and inserts!
  - What about unique indexes?

Parallel Aggregates

❖ For each aggregate function, need a decomposition:
  - \( \text{count}(S) = \sum \text{count}(s(i)) \)
  - \( \text{avg}(S) = \left( \sum \text{sum}(s(i)) \right) / \sum \text{count}(s(i)) \)
  - and so on...
❖ For groups:
  - Sub-aggregate groups close to the source.
  - Pass each sub-aggregate to its group’s site.
  - Chosen via a hash fn.
### Parallel Joins

- Nested loop:
  - Each outer tuple must be compared with each inner tuple that might join.
  - Easy for range partitioning on join cols, hard otherwise!
- Sort-Merge (or plain Merge-Join):
  - Sorting gives range-partitioning.
  - But what about handling 2 skews?
  - Merging partitioned tables is local.

### Parallel Hash Join

- In first phase, partitions get distributed to different sites:
  - A good hash function automatically distributes work evenly!
- Do second phase at each site.
- Almost always the winner for equi-join.

### Dataflow Network for \(|\mid\) Join

- Good use of split/merge makes it easier to build parallel versions of sequential join code.

### Complex Parallel Query Plans

- Complex Queries: Inter-Operator parallelism
  - Pipelining between operators:
    - note that sort and phase 1 of hash-join block the pipeline!!
  - Bushy Trees

### Observations

- It is relatively easy to build a fast parallel query executor
  - S.M.O.P.
- It is hard to write a robust and world-class parallel query optimizer.
  - There are many tricks.
  - One quickly hits the complexity barrier.
  - Still open research!
Parallel Query Optimization

- Common approach: 2 phases
  - Pick best sequential plan (System R algorithm)
  - Pick degree of parallelism based on current system parameters.
- "Bind" operators to processors
  - Take query tree, "decorate" as in previous picture.

What’s Wrong With That?

- Best serial plan ≠ Best || plan! Why?
- Trivial counter-example:
  - Table partitioned with local secondary index at two nodes
  - Range query: all of node 1 and 1% of node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index.
- SELECT *
  FROM telephone_book
  WHERE name < “NoGood”;

Parallel DBMS Summary

- ||-ism natural to query processing:
  - Both pipeline and partition ||-ism!
- Shared-Nothing vs. Shared-Mem
  - Shared-disk too, but less standard
  - Shared-mem easy, costly. Doesn’t scaleup.
  - Shared-nothing cheap, scales well, harder to implement.
- Intra-op, Inter-op, & Inter-query ||-ism all possible.

DBMS Summary, cont.

- Hardest part of the equation: optimization.
  - 2-phase optimization simplest, but can be ineffective.
  - More complex schemes still at the research stage.
- We haven’t said anything about Xacts, logging.
  - Easy in shared-memory architecture.
  - Takes some care in shared-nothing.

DBMS Summary, cont.

- Data layout choices important!
- Most DB operations can be done partition-||
  - Sort.
  - Sort-merge join, hash-join.
- Complex plans.
  - Allow for pipeline-||, but sorts, hashes block the pipeline.
  - Partition ||-ism achieved via bushy trees.