Parallel DBMS

CMPSCI 645

Slide content due to Ramakrishnan, Gehrke, Hellerstein, Gray.

Parallel vs. Distributed DBs

- Parallel database systems
  - Improve performance through parallelizing various operations: loading data, indexing, query evaluation. Data may be distributed, but purely for performance reasons.

- Distributed database systems
  - Data is physically stored across various sites, each of which runs DBMS and can function independently. Data distribution determined by local ownership and availability, in addition to performance.
Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

1,000 x parallel
1.5 minute to scan

Bandwidth

1 Terabyte
10 MB/s

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

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

- **Speed-Up**
  - More resources means proportionally less time for given amount of data.
  - Problem size constant, system grows.

- **Scale-Up**
  - If resources increased in proportion to increase in data size, time is constant.
  - Problems size, system both grow
Enemies of good speed-up / scale-up

- **Start up work**
  - If thousands of processes must be started, this can dominate actual computation time

- **Interference**
  - The slowdown each new process imposes on all others when accessing shared resources

- **Skew**
  - Variance in the size of jobs for each process. Service time for whole job is the service time of **slowest** step of job.
Architecture Issue: Shared What?

- Alternative architectures:
  - Shared memory: all processors shared common global memory and access to all disks.
  - Shared disk: all processors have private memory, but direct access to all disks.
  - Shared nothing: each memory/disk owned by processor which acts as server for data.
Architecture Issue: Shared What?

- Alternative architectures:
  - Shared memory: all processors share common global memory and access to all disks.
  - Shared disk: all processors have private memory, but direct access to all disks.
  - Shared nothing: each memory/disk owned by processor which acts as server for data.

Advantages:
- Minimize interference by minimizing shared resources
- Exploit commodity processors and memory
- Disk and memory accesses are local
- Traffic on interconnection network is minimized

- Shared nothing: each memory/disk owned by processor which acts as server for data.
Different Types of DBMS | -ism

- 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 -ism
Limits of pipelined parallelism in DBMS

- Relational pipelines usually not very long
- Some relational operators block (e.g. sorting, aggregation)
- Execution cost of one operator may be much higher than another (example of skew)

- As a result, partitioned parallelism is key to achieving speed-up and scale-up
Automatic Data Partitioning

Partitioning a table:

**Range**
- Good for equijoins, range queries, group-by

**Hash**
- Good for equijoins

**Round Robin**
- Good to spread load

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning
Parallel query processing

Two relational scans consuming two input relations, A and B, and feeding their outputs to a join operator that in turn produces a data stream C.
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.
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.

**Diagram:**
- **Phase 1**
  - Original Relations (R then S)
  - Disk
  - Input to hash function $h$
  - $B$ main memory buffers
  - Output partitions
  - Disk
Dataflow Network for \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

```
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

```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
Parallel query optimization issues

- Cost estimation in parallel environment
- Consider bushy plans -- much larger plan space
- Some parameters only known at runtime: number of free processors, available buffer space.
Best serial plan != Best | | plan!

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.
Data layout choices important!

Most DB operations can be done partition-

- Sort.
- Sort-merge join, hash-join.

Complex plans.
- Allow for pipeline-ism, but sorts, hashes block the pipeline.
- Partition-ism achieved via bushy trees.
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.