

CMSC724: Parallel/Distributed Databases*

Amol Deshpande

March 11, 2008

Database Machines

- Proposals in late 70's, early 80's for specialized hardware
 - Database Machines: An idea whose time has passed ?
 - * Boral, DeWitt, 1983
 - Processor-per-track:
 - * Database specific storage
 - * Evaluate selections directly on the CPs etc...
 - Processor-per-head
 - * Need parallel readout
 - * Combined with indexes, gives good performance
 - Off-the-track
 - * Something like a shared-memory machine
 - * Used special DB-specific processors

Database Machines

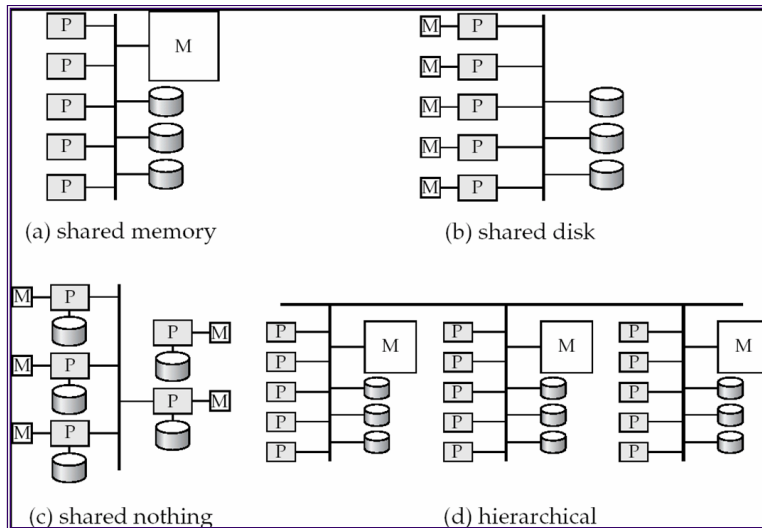
- Didn't work
 - Processor-per-track:
 - * Not cost-effective
 - * Based on fixed-head disks, or solid-state storage
 - Processor-per-head
 - * Parallel readouts are hard to do ??
 - Off-the-track
 - * Disk bandwidth is actually decreasing ??? (maybe true then)
 - Generally, specialized hardware is hard to make work

*Based on notes from Joe Hellerstein

- * Too expensive, slow-to-evolve, requires a tool set
- * Doesn't help too much with sorts/joins anyway
- * General-purpose hardware improves faster
- * Soon catches up

- IDISKS ? (late 90's)

Types of Parallelism



Types of Parallelism

- Shared memory
 - One of the last remaining “cash cows”
 - Direct mapping from uni-processor
 - Data structures shared between processors
 - * Process models extend naturally: processes or threads assigned to different processes
 - * Cache-coherency can be issues: typically left to the hardware
 - Resurgence as **multi-core**
 - * Separate caches, but usually not large enough to call shared-nothing
- Shared disk
 - Increasingly common because of SAN (storage area network)
 - Better failure behavior (since data still available)
 - Distributed lock managers, cache-coherency etc...

Types of Parallelism

- Shared nothing
 - Perhaps most common and most scalable
 - Horizontal data partitioning
 - * Good partitioning schemes essential
 - * More burden on the DBA
 - Query processing and optimization challenging
 - Partial failures
 - * Option 1: Can skip the data on the machine that failed
 - * Option 2: Bring down the whole system (“Data skip”)
 - * Option 3: Redundancy (usually “hot standby”)
- NUMA (Non-uniform Memory Architecture)
 - Processors have different access costs for different parts of memory
 - Option 1: ignore non-uniformity (treat as shared-memory)
 - Option 2: minimize cross-processor access to memory (treat as shared-nothing/shared-disk)

Concepts...

- Database operations “embarrassingly parallel”
- Speedup vs Scaleup
 - Speedup: old time/new time
 - Scaleup: how many more queries/how much larger query can you solve
- Types of parallelism:
 - Pipelined
 - * Each “operator” on a different processor
 - * Easier to setup, but low parallelism
 - Partitioned
 - * Split relations horizontally, replicate the operators
 - * Exploits all processors, but much harder to setup
 - * Optimization messy: Need to make decisions about how to split etc..

Concepts...

- Storage: Round-robin vs Hash-based vs Range-based
 - Bubba used “heat” to partition
- Barriers to linearity
 - Startup overheads (remember **Amdahl's law**)
 - Interference
 - * Communication overhead, waiting on queues etc..
 - * If the interference just 1%, the maximum speedup < 37
 - Skew
 - * Partitioning may turn out to be non-uniform (common cause: duplicates)
 - * Solution 1: Carefully design hash functions
 - * Solution 2: Use a very fine-grained partitioning function, and adjust the assignment of partitions to processors

Concepts...

- Autonomy ?
 - Not autonomous, centralized decision-making
- Concurrency/locking ?
 - Two-phase locking (2PL)
 - Probably two-phase commit (2PC)
 - Centralized deadlock detection
- Recovery
 - ARIES-based (similar to centralized)
- Failures ?
 - Chained declustering
 - * Each relation partition replicated on one other site
 - * Many similarities to RAID

Query execution

- New operators: Hash joins, replicate-all strategy etc...
- Left-deep trees good for pipelining
 - Beware: Some people (eg. DeWitt) calls these “right-deep”
- Selections: Indexes etc (individual at each site)
- Joins: Hash joins, replicate-all
 - Symmetric hash join operator
- Sorting
 - Sort partitions in parallel, merge is computationally trivial
- Aggregation: Do separately, and combine
 - Can all aggregates be done like this ?

Engineering issues

- Re-use existing code
- Gamma: Split/merge operators
- Volcano:
 - Exchange operator
 - Allows arbitrary interleavings
 - An operator can directly call another operator (within the process), across processes or across network
 - Data-driven vs Demand-driven dataflows (pull vs push)
 - Semaphores used for controlling producer vs consumer rates (**flow control**)
 - Also, rule-based extensible optimizer (became the basis for MSSQL Server Optimizer framework)
 - [Later work by Mehul Shah on extending Exchange](#)

Query optimization

- Much larger plan space
 - Need to worry about partitioning, different indexes at different sites..
- Cost metric:
 - Communication cost ?
 - Response time is not a nice metric
 - * Conflicts with traditional **total work** metric
 - * May prefer to optimize for total work, and handle more queries instead
- 2-phase optimization (XPRS)
 - Phase 1: Optimize for total work
 - Phase 2: Parallelize the plan
- Load balancing/skew: Recursive partitioning for hash joins

Distributed Databases

- R* etc...
- Communication costs much higher
 - Use semi-joins/bloom filters etc
- More autonomy per machine
 - Typically different administrative domains
 - Different schemas, even different machines
- Federated ?
 - Mariposa – Used the economic paradigm
 - For query processing, replication etc.