Advanced Topics in Computer Systems, CS262B

Prof Eric A. Brewer

Parallel Databases

February 17, 2004

I. The Future or Parallel Database Systems

[based on notes from Joe Hellerstein]

Parallelism research happened along multiple tracks. OS/Compilers/Scientific community was one track in the 80's. Parallel DBMS was another. It's worth noting that mainframes were parallel computers long before it was sexy...though the parallelism was used mostly just for multitasking.

A pattern in parallel systems research?

1. Explore specialized hardware to give better performance through parallelism
2. Move parallelism ideas into software running over commodity hardware
3. Along the way, better understanding of algorithms, pounding down of architectural bottlenecks
4. Drive for performance and scale leads to reliability/maintainability research

The database machine

(Boral/DeWitt '83, "Database Machines: An Idea Whose Time has Passed?")

All mixed up: trying to make storage devices faster, and do "more of the work". Something of a hodge-podge of extra processors and novel storage devices, and combinations of the two.

- o Processor Per Track (PPT): Examples: CASSM, RAP, RARES. Goal: each processor sees random-access storage, no seeks or indexes (super-parallelism). Too expensive. Wait for bubble memory, charge-couple devices (CCDs)? Still waiting...
- o PP Head: Examples: DBC, SURE. Helps with selection queries. Avoids the transfer of data across the I/O channel. Also cylinder-per-revolution for maximum parallelism -- parallel readout disks (read all heads at once). This has become increasingly difficult over time ("settle" is increasingly tricky as disks get faster and smaller).
- o Off-the-disk DB Machines: Examples: DIRECT, RAP.2, RDBM, DBMAC, INFOPLEX. Precursors of today's parallel DBs. DIRECT: A controller CPU, and special-purpose query processing CPUs with shared disks and memory (shared everything!).
    - 1981: Britton-Lee Database Machine. A Sun box with a special processor. They had their own OS and compiler for that processor.
    - "There was 1 6Mhz Z8000 Database processor, up to 4 Z8000 communications processors. Up to 4 "Intelligent" disk controllers (they could optimize their own sector schedule, I think). I think you could put up to 16.5Gig drives on the thing and probably 4-6 MB of memory. About a year later we upped the DBP to 10Mhz (yielding a mighty 1 mips). " -- Mike Ubell

All failed. Why?

- o these don't help much with sort, join, etc. Only with easy stuff. Not clear whether it addressed the performance bottlenecks that existed in the hardware of the time -- even less clear today.
- o Important lesson: special-purpose hardware is a losing proposition
  - prohibitively expensive (no economy of scale)
  - slow to evolve
  - requires a tool set

Performance metrics:

* Speedup x=old_time/new_time.

* Scaleup. small_sys_elapsed_small_prob/big_sys_elapse_big_prov

o Transaction scaleup: N times as many TPC-C's for N machines

o Batch scaleup: N times as big a query for N machines

2 kinds of data parallelism

o pipelined (most are short in traditional QP)

o partition

3 barriers to linearity:

- o startup overheads
- o interference: usually the result of unpredictable communication delays (comm cost, empty pipelines)
- o skew

3 basic architectures.

o shared-memory

o shared-disk

o shared-nothing

Ask yourself about:

1. ease of programming
2. cost of equipment (and size of its user base)

3. reliability/availability (both MTTF and MTTR)

4. who controls resources and how

5. performance goals: esp. latency vs. bandwidth. Where's your system bottleneck?

6. maintenance: utilities, DB design, admin wizardry

For an entertaining early take on all this, see Stonebraker's quickie overview "The Case for Shared Nothing" (HPTS '85). Interestingly, doesn't cover the "batch scaleup" problem (probably because of the charter of the HPTS workshop)

DeWitt et al., GAMMA DB Machine

* Gamma: "shared-nothing" multiprocessor system

* Other shared-nothing systems: Bubba (MCC), Volcano (Colorado), Teradata (now owned by NCR), Tandem, Informix, IBM DB2 Parallel Edition

* Gamma Version 1.0 (1985)

    o shared nothing: token ring connecting 20 VAX 11/750's.

    o Eight processors had a disk each.

    o Architectural tuning issues:

        + Token Ring packet size was 2K, so they used 2K disk blocks. Mistake.

        + Bottleneck at Unibus (1/20 the bandwidth of the network itself, slower than disk!)

        + Network interface also a bottleneck – only buffered 2 incoming packets

        + Only 2M RAM per processor, no virtual memory

        + This becomes a pattern in the literature: architectural system balance

* Version 2.0 (1989)

    o Intel iPSC-2 hypercube. (32 386s, 8M RAM each, 1 333M disk each.)

    o networking provides 8 full duplex reliable channels at a time

    o small messages are datagrams, larger ones form a virtual circuit which goes away at EOT

    o Usual multiprocessor story: tuned for scientific apps

        + OS supports few heavyweight processes

        + Solution: write a new OS! (NOSE) – more like a thread package

        + SCSI controller transferred only 1K blocks

    o Other complaints

+ want disk-to-memory DMA. As of now, 10% of cycles wasted copying from I/O buffer, and CPU constantly interrupted to do so (13 times per 8K block!)

* How does this differ from a distributed dbms?
    - o no notion of site autonomy
    - o centralized schema
    - o all queries start at "host"
    - o assumption of very high bandwidth
* Storage organization: all relations are "horizontally partitioned" across all disk drives in four ways:
    - o round robin: default, used for all query outputs
    - o hashed: randomize on key attributes
    - o range partitioned (specified distribution), stored in range table for relation
    - o range partitioned (uniform distribution – sort and then cut into equal pieces.)
    - o Within a site, store however you wish.
    - o indices created at all sites
    - o A better idea: heat (Bubba?)
        - + hotter relations get partitioned across more sites
        - + why is this better?
    - o Note: all this adds to the administration of a DBMS. Yuck.
* "primary multiprocessor index" identifies where a tuple resides.

Concurrency and Recovery

* CC: 2PL with bi-granularity locking (file & page). Centralized deadlock detection. (Another distinction with distributed)
* ARIES-based recovery, static assignment of processors to log sites.

Availability, Fault Tolerance: Chained Declustering

* nodes belong to relation clusters
* relations are declustered within a relation cluster
* backups are declustered "one disk off"
* tolerates failure of a single disk or processor

* discussion in paper vs. interleaved declustering
  - o these kinds of coding issues have been beaten to death since
  - o note that RAID does this for files (sequences), declustering does it for relations (sets). Sets are much nicer to work with.
* glosses over how indexes are handled in case of failure (another paper)

Performance Results

* Simplified Picture: Gamma gets performance by...
  - o Running multiple small queries in parallel at (hopefully) disjoint sites.
  - o Running big queries over multiple sites --- idea here is:
    - + Logically partition problem so that each subproblem is independent, and
    - + Run the partitions in parallel, one per processor.
      - # Examples: hash joins, sorting, ...

* Performance results:
  - o a side-benefit of declustering is that small relations mean fewer & smaller seeks (?? WHAT??)
  - o big queries get linear speedup
    - + not perfect, but amazingly close
  - o pretty constant scaleup
  - o some other observations
    - + hash-join goes a bit faster if already partitioned on join attribute
      - # but not much! Redistribution of tuples isn't too expensive -- bottleneck not in comm.
    - + as you add processors, you lose benefits of short-circuited messages, in addition to incurring slight overhead for the additional processes

Missing Research Issues (biggies!):

* query optimization (scheduling, query rewriting for subqueries)
* load balancing: inter-query parallelism with intra-query parallelism
* disk striping, reliability, etc.
* online admin utilities

* database design

* skew handling for non-standard data types

Some Themes in Parallel DBs (that distinguish them from other parallel programming tasks):

- o Hooray for the relational model
  - apps don't change when you parallelize system (physical data independence!). can tune, scale system without informing apps too
  - ability to partition records arbitrarily, w/o synchronization
  - lack of pointers means no need for low-latency transfer of data
  - instead of pointer-chasing, batch partitioning + joins.....THIS IS GENERALIZABLE!
- o essentially no synchronization except setup & teardown
  - no barriers, cache coherence, etc.
  - DB transactions work fine in parallel

  + data updated in place, with 2-phase locking transactions

  + replicas managed only at EOT via 2-phase commit

  + coarser grain, higher overhead than cache coherency stuff
- o Bandwidth much more important than latency
  - often pump 1-1/n % of a table through the network
  - aggregate net BW should match aggregate disk BW
  - bus BW should match about 3x disk BW (NW send, NW receive, disk)
  - Latency, schmatency. Insignificance makes a BIG difference in what architectures are needed.
- o Shared mem helps with skew
  - but distributed work queues can solve this (?) (River)