Parallel Query Processing

Berkeley cs186

Why Parallelism?

- At 3Gbit/sec = 375MB/sec
  - Scan 100TB in ~280,000 sec = ~3.25 days

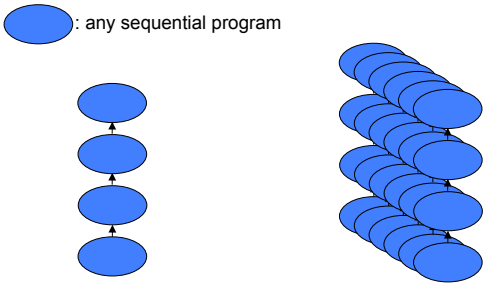
Why Parallelism?

- At 3Gbit/sec = 375MB/sec
  - Scan 100TB in ~280,000 sec = ~3.25 days
- Run it 100-way parallel
  - Scan 100TB in ~45 minutes
- 1 big problem = many small problems!
  - Trick: make them independent

A little history

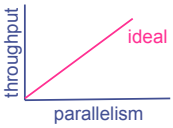
- Relational revolution
  - declarative set-oriented primitives
  - 1970's
- Parallel relational database systems
  - on commodity hardware
  - 1980's
- Renaissance: MapReduce etc.
  - now

Two Kinds of Parallelism

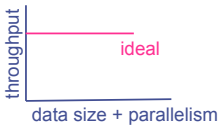


Two Kinds of Benefit

- Speed-up
  - Increase HW
  - Fix workload



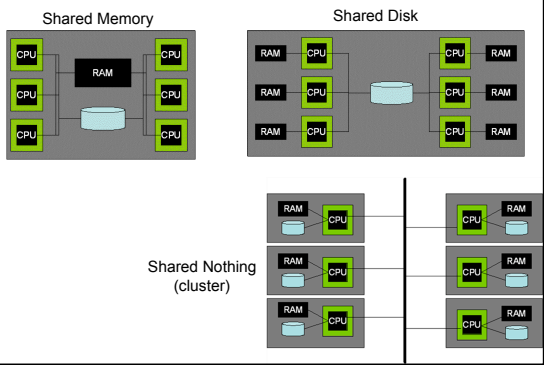
- Scale-up
  - Increase HW
  - Increase workload



"Big Data" is GREAT for Parallelism!

- Why?
  - Set-oriented languages
  - Batch operations
  - Pre-existing divide-and-conquer algorithms
  - Natural pipelining

Parallel Architectures



Some Early Systems

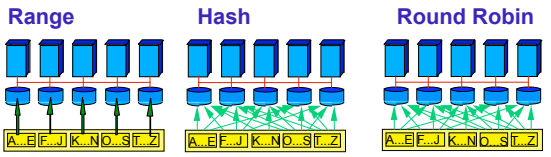
- Research
  - XPRS (Berkeley, shared-memory)
  - Gamma (Wisconsin, shared-nothing)
  - Volcano (Colorado, shared-nothing)
  - Bubba (MCC, shared-nothing)
- Industry
  - Teradata (shared-nothing)
  - Tandem Non-Stop SQL (shared-nothing)

Uses of Parallelism

- Inter-query
  - Esp. for Transaction Processing
    - Wait for discussion of Concurrency Control
- Intra-query
  - Inter-operator
    - Tree
    - Pipeline
  - Intra-operator
    - Divide & Conquer
    - Focus here – best bang for the buck

Data Partitioning

Partitioning a table:



Good for equijoins, range queries group-by

Good for equijoins, group-by

Good for spreading load

Shared nothing benefits from "good" partitioning
Remind you of something?

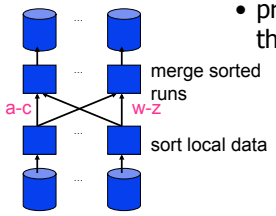
Parallel Scans

- Scan in parallel, merge (concat) output
- Selection: may skip sites in some cases
  - 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 (keys)?

Lookup by key

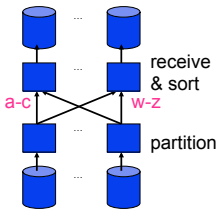
- data partitioned on function of key?
  - great!
- otherwise
  - umm…

Sort



- problems with this approach?

Sort, improved



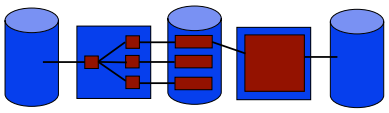
- a key issue: avoiding skew
  - sample to estimate data distribution
  - choose ranges to get uniformity

Sorting Records!

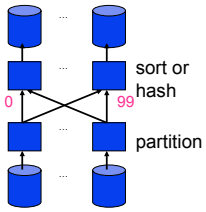
- Sorting has become a blood sport!
  - Parallel sorting is the name of the game …
- Minute Sort: how many 100-byte records can you sort in a minute?
  - Current World record: 264 ***GB***
    - 400 x (dual-3.2GHz processors, 6 disks, 8GB RAM) (2007)
- Penny Sort: how many can you sort for a penny?
  - Current world record: 190 GB
    - 2408 seconds on a $393 Linux/AMD system (2008)
    - $393 spread over 3 years worth of seconds = 2408 seconds/penny
- Joule Sort: how many can you sort with a Joule?
- See http://www.hpl.hp.com/hosted/sortbenchmark/

Parallel Hashing

- Well … think about single-site hashing
  - Phase 1: partition input
  - Phase 2: build in-memory hashtable
- Where do you put the communication?
  - And how do you split data across sites?



Grouping

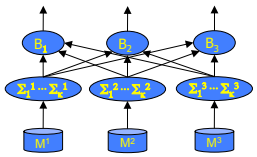


- again, skew is an issue
- approaches:
  - **avoid** (choose partition function carefully)
  - **react** (migrate groups to balance load)

Parallel Aggregates/GroupBy

- For each aggregate function, need a local/global decomposition:
  - **sum**(S) = **Σ Σ** (s)
  - **count** = **Σ count** (s)
  - **avg**(S) = (**Σ Σ** (s)) / **Σ count** (s)
  - etc…
- Group by:
  - Do a local aggregate for each group
    - 3 machines ($M^1..M^3$), k "dno" sums each
  - Ship each local agg to an appointed "collector" to compute global agg
    - Hash function maps "dno" to one of 3 collector "buckets" ($B_1..B_3$)
    - Sum up each group at exactly 1 collector

SELECT dno, SUM(sales)
FROM all_sales
GROUP BY dno



Advanced topic: can you think of a summary statistic this doesn't work for?

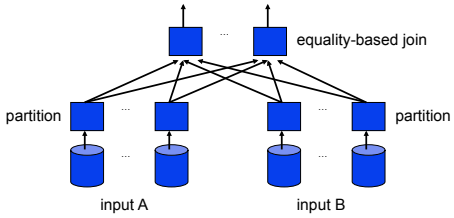
Parallel Joins

- Nested loop:
  - Compare each outer tuple with each inner tuple that might join.
  - Easy for range/hash partitioning and equijoin
    - Hard otherwise!
- Sort-Merge (or plain Merge-Join):
  - Sorting gives range-partitioning.
    - But what about handling 2 skews?
  - Merging partitioned tables is local.
- Hash-join
  - Hash-partition both inputs
  - build/probe phase is local

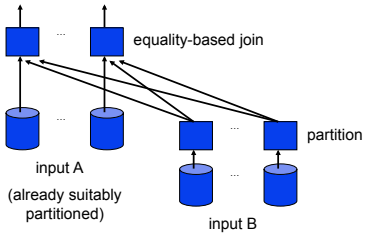
Joins: Bigger picture

- alternatives:
  - symmetric repartitioning
    - what we did so far
  - asymmetric repartitioning
  - fragment and replicate
  - generalized f-and-r

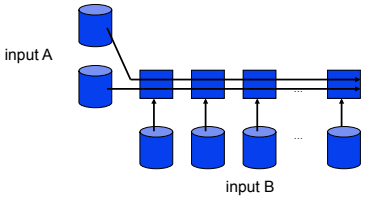
join: symmetric repartitioning



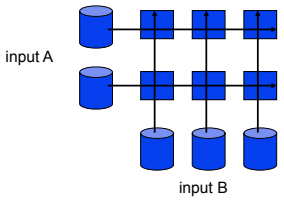
join: asymmetric repartitioning



join: fragment and replicate

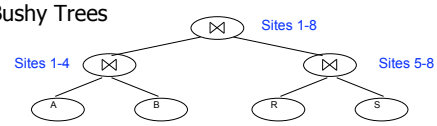


join: generalized f-and-r

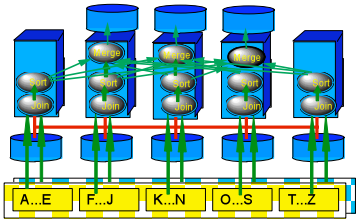


Complex Parallel Query Plans

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



The Whole Enchilada



N inputs, M outputs, no bottlenecks.

Partitioned Data
Partitioned and Pipelined Data Flows

MapReduce @ Google

- Shared-disk (shared filesystem)
- Goals
  - Familiar programming (Java) over files
    - More on this in a few weeks
  - Auto-parallelization
  - Simple load-balancing and "fault tolerance" on LOTS of machines
    - Why is this important as you increase the value of LOTS
- Main trick: no pipelining
  - Always store iterator outputs to disk
    - A "master" node can keep track of iterator partitions that fail and restart them on the relevant inputs
  - Worth it? Hmm….

Parallel DBMS Summary

- parallelism natural to query processing:
  - Both pipeline and partition
- Shared-Nothing vs. Shared-Mem vs. Shared Disk
  - Shared-mem easiest SW, costliest HW.
    - Doesn't scale.
  - Shared-nothing cheap, scales well, harder to implement.
  - Shared disk a middle ground
    - Introduces icky stuff related to concurrency control
- Intra-op, Inter-op, & Inter-query parallelism all possible.

Parallel DBMS Summary, cont.

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

Parallel DBMS Summary, cont.

- Hardest part of the equation: query optimization.
  - Wait for it!
- We haven't said anything about Xacts, logging.
  - Familiar in shared-memory architecture.
  - Takes some care in shared-nothing.
  - Yet more tricky in shared-disk