Parallel DBMS

Prof. Yanlei Diao
University of Massachusetts Amherst

Slides Courtesy of R. Ramakrishnan and J. Gehrke
I. Parallel Databases 101

• Rise of parallel databases: late 80’s
• Architecture: shared-nothing systems
  – A number of nodes connected by fast Ethernet switches
  – But used special-purpose hardware (costly, slow to evolve)
• Typical systems
  – Gamma: U. of Wisconsin Madison
  – TeraData: Wal-Mart’s 7.5TB sales data in hundreds of machines
  – Tandem
  – IBM / DB2
  – Informix…
Some “||” Terminology

- **Speed-Up**
  - Hold the problem size, grow the system
  - Report *serial time/||-time*
  - Ideally, linear

- **Scale-Up**
  - Grow both the system and the problem, report running time
  - Report *serial time/||-time*
  - Ideally, constant
Barriers to Linear Speedup/Scaleup

- **Startup overheads**
  - the time needed to start a parallel operation.

- **Interference**
  - the slowdown that each new process imposes on all others when accessing shared resources.

- **Skew**
  - As the number of parallel tasks increases, the average size of each task decreases, but the variance can exceed the mean. So the service time of a job is determined by the slowest step.
  - For example, consider partitioned parallelism and hot spots of data access.
Different Types of DBMS  | | -ism

- **Partitioned parallelism**
  - Partition data over all nodes, get the nodes working to compute a given operation (scan, sort, join)

- **Pipelined parallelism**
  - A chain of operators O_1, O_2, ..., O_k run in parallel, with O_1 working on tuple t_n, O2 on t_(n-1), ... O_k on t_(n-k+1)
  - Can run these operators on different nodes
  - Some operators break pipelining, e.g. sort, hash

- **Independent operators**
  - Consider bushy query plans
  - A join B, C join D are independent

- We’ll focus on partitioned  |  | -ism
Data Partitioning Schemes

Partitioning a table:

**Range**
- Good for seq. scan, associative search, sorting
- Can have data skew

**Hash**
- Good for seq. scan, equality search, equijoins if they match the hash attr
- Bad for range search, or operations that do not match the hash attr;
  Can also have data skew

**Round Robin**
- Good for seq. scan
- Useless for other query operations
Query Execution Architecture

Fig. 2. Gamma process structure.
(1) Parallel Scan and Associative Access

- **Scan**: do it in parallel, and merge.

- **Associative Access**: Selection may not require all sites for range or hash partitioning.
  - Want to restrict selection to a few nodes, or restrict “small” queries to a few nodes.
  - Indexes can be built at each partition.
  - What happens during data inserts and lookups?
(2) Parallel Sorting

- **Parallel Sorting:**
  - Scan in parallel, and range-partition as you go.
  - As tuples come into each node, begin “local” sorting.
  - Resulting data is sorted, and range-partitioned.
  - Problem: *skew!*
  - Solution: “sample” the data at start to determine partition points.

Some record in the history:
8.5 Gb/minute, shared-nothing; Datamation benchmark in 2.41 secs
(UCB students! http://now.cs.berkeley.edu/NowSort/)
(3) Equi-Join

- **Equi-joins**: partition the two input relations across all nodes, and compute the join locally at each processor.

- Can use either *range partitioning* or *hash partitioning*, on the join attribute
  - \( R \) and \( S \) each are partitioned into \( n \) partitions, denoted \( R_0, R_1, \ldots, R_{n-1} \) and \( S_0, S_1, \ldots, S_{n-1} \).
  - Partitions \( R_i \) and \( S_i \) are sent to node \( i \).
  - Each node locally computes the join using any method.
Two-Phase Hash Join (Grace 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.
Parallel Two-Phase Hash Join

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

1. Run the analysis of hybrid hash join as before, but using the aggregate memory $B^*$, to determine the number of logical buckets $N$; hash fn $h_1$ maps tuples to buckets.

2. The in-memory join, $R_0 \bowtie S_0$, is partitioned over $M$ processors using hash fn $h_2$.

3. Disk resident buckets, $R_i$ and $S_i$ ($i>0$), are partitioned over $K$ disks using $h_3$.

4. For each subsequent bucket $i$ ($i>0$), partition $R \bowtie S_i$ over $M$ processors using $h_2$.

Yanlei Diao, University of Massachusetts Amherst
Parallel Hybrid Hash Join

Yanlei Diao, University of Massachusetts Amherst
• **Non-equijoins**: partitioning not possible for some join conditions
  - E.g., non-equijoin conditions, such as R.A > S.B.

• Use the **fragment and replicate** technique
  - 1) Special case: only one relation is partitioned
    - R is partitioned; any partitioning technique can be used.
    - The other relation, S, is replicated across all the nodes.
    - Node i then locally computes the join of \( R_i \) with all of S using any join technique.
    - Works well when S is small.
  - 2) General case: both R and S are partitioned
    - Need to replicate all \( R \) partitions or all \( S \) partitions
    - Depicted on next slide
Illustrating Fragment-and-Replicate Join

(a) Asymmetric fragment and replicate

(b) Fragment and replicate
(5) Parallel Aggregates

• **Aggregates**: for each aggr function, need a decomposition:
  – **Distributive**: $\text{count}(S) = \sum \text{count}(s(i))$, same for $\text{sum}()$
  – **Algebraic**: $\text{avg}(S) = (\sum \text{sum}(s(i))) / \sum \text{count}(s(i))$
  – **Holistic**: e.g., median, quantiles

• For group-by aggregation:
  – How would you implement parallel group by?
  – How do you add aggregation to each group?
Volcano Database

- **The iterator model**: each operation is implemented by 3 functions:
  - Open(): sets up the data structures and performs initializations
  - GetNext(): returns the next tuple of the result.
  - Close(): ends the operations. Cleans up the data structures.
- Enables pipelining!
- A **pull based model**, which can minimize the amount of intermediate storage.
- Contrast with **data-driven materialize model**.
  - Sometimes it’s the same (e.g., sorted scan).

_Yanlei Diao, University of Massachusetts Amherst_
Volcano Parallel Database

• Volcano parallel database popularizes the *exchange-operator* model
  - exchange operator is introduced into query plans to partition and distribute tuples
  - each operation works independently on local data on each processor, in parallel with other copies of the operation

• One architecture to support all three forms of parallelism
  - Independent parallelism
  - Pipelined (vertical) parallelism
  - Partitioned (horizontal) parallelism: see the example in Fig. 3
What is missing: Parallel Query Optimization

- Common approach: 2 phases
  - Pick best single-machine 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 parallel 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”;

```sql
SELECT *
FROM telephone_book
WHERE name < "NoGood";
```
What is missing, cont

• Fault tolerance
  – Suppose that each node fails with probability $p$
  – What is the chance that a job fails with $n$ nodes?
  – Parallel DBMS will restart a failed job.

• Use of special, expensive hardware
  – Hard to afford
  – How to upgrade or port the software

• ...