

Parallel Database Systems: The Future of High Performance Database Processing

David J. DeWitt and Jim Gray

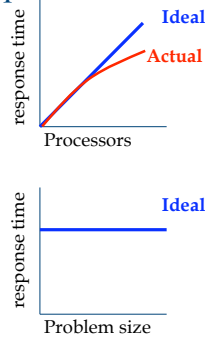
Presentation: Kristen LeFevre

Introduction

- Parallel databases an early success story in parallel processing
 - Many commercial systems available today
- Why?
 - Relational (“dataflow”) operators applied to uniform streams of data
 - SQL language can be parallelized (old software doesn’t need to be rewritten)
 - Users / apps don’t need to think in parallel

Speedup and Scaleup Goals

- **Linear Speedup:** Twice as much hardware performs a task twice as fast
- **Linear Scaleup:** Twice as much hardware performs twice as big a task in the same time



Speedup and Scaleup

- General barriers to linear speedup and scaleup
 - Startup: Time to start a parallel process
 - Interference: Slowdown caused by access to shared resources
 - Skew: High variance in the performance of parallel jobs

Hardware Architectures

- **Ideal machine:** Infinitely fast processor, infinite memory, infinite bandwidth, free
- **Goal:** “... build an infinitely fast processor out of infinitely many processors of finite speed, and to build an infinitely large memory with infinite bandwidth from infinitely many storage units of finite speed”

Hardware Architectures

- **Three main alternatives**

DB	OS / Architecture
[Stonebraker86] Shared Disk	Network attached storage
Shared Memory	Symmetric multiprocessor (SMP)
Shared Nothing	Cluster

Hardware Architectures

- **Shared Disk**

Main disadvantage:
All I/O has to cross interconnect

Hardware Architectures

- **Shared Memory**

Main disadvantage:
Expensive

Hardware Architectures

- **Shared Nothing**

Big Advantages:
No data access interconnect, processors don't interfere with one another

Commodity PC

Parallelizing SQL

- Two main kinds of parallelism
- **Pipelined**
 - Stream results of one relational operator into another
 - Limited degree (I.e., limited number of pipelines operations in a query)
- **Partitioned**
 - Partition the data across the disks in the system
 - Run one operation as a set of smaller ops
 - Fits naturally into the shared-nothing architecture

Parallelizing SQL

- **Horizontal Data Partitioning**
 - Distribute tuples across multiple disks
- E.g., Divide the Students relation across N disks

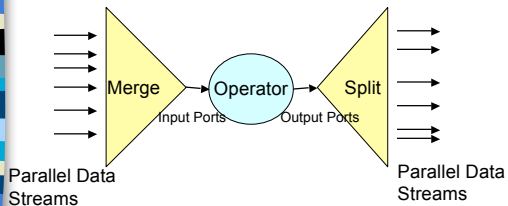
How to Partition?

Data Skew: Too much Data on too few disks

- **Round Robin Partitioning**
 - Spread tuples evenly across disks in a round-robin fashion
 - Always leads to even spread of data
- **Hash Partitioning**
 - Partitioning attribute A
 - $H(\text{tuple}.A) \rightarrow [1, N]$ (# of disks)
 - Convenient way of known which disk a tuple is on
 - Risks data skew
- **Range Partitioning**
 - Partitioning Attribute A
 - Assign non-overlapping ranges to each disk
 - Good for range-lookup
 - Risks data skew

Parallelizing SQL

- How to execute a SQL query in parallel on top of horizontally partitioned data?
- Basic idea is simple: Split and Merge Ops

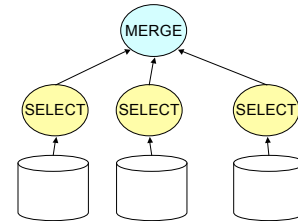


Parallelizing SQL

- Simple Example

```
SELECT *
FROM Students
WHERE Age > 22
```

Assume Students
Is RR-partitioned

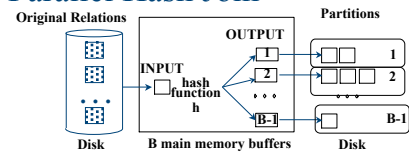


Use SPLIT if we want to feed the resulting stream into another parallel operator (e.g., a join)

Parallel Joins

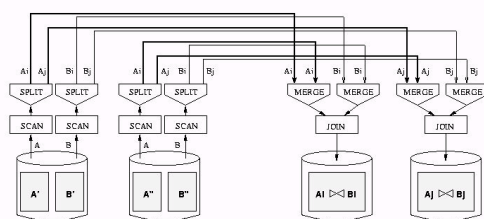
- Nested loop:
 - Each outer tuple must be compared with each inner tuple that might join.
 - Easy for range (also hash for equijoins) partitioning on join columns, hard otherwise!
- Sort-Merge:
 - Sorting gives range-partitioning.
 - Merging partitioned tables is local.

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.

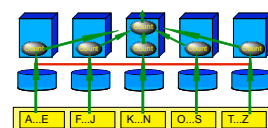
Parallel Join



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

Parallel Aggregates

- For each aggregate function, need a decomposition:
 - $\text{count}(S) = \sum \text{count}(s(i))$, ditto for $\text{sum}()$
 - $\text{avg}(S) = (\sum \text{sum}(s(i))) / \sum \text{count}(s(i))$
 - and so on...
- Each processor computes partial result on its partition; combine partial results at a single site





Summary

- Parallel Databases / Parallel SQL an early success story
- Modern systems architectures based on shared-nothing (cluster) design
- Existing relational operators easily parallelized by using data partitioning
- Specialized operators, too