
Parallel DBMSs

Christos Faloutsos
www.cs.cmu.edu/~christos

Roadmap

1) RDB: System R and Ingres
2) Implementation: buffering, indexing, q-opt
3) Transactions: locking, recovery
4) Distributed DBMS
5) Parallel DBMSs: Gamma, Alphasort
6) OO/OR DBMS
7) Data Analysis - data mining
8) Benchmarks
9) vision statements
   extras (streams/sensors, graphs, multimedia, web, fractals)

Citation


Main Message

• Technology trends give
  – Many processors and storage units
  – Inexpensively
• Parallelism in DBs came from a failed idea
  – (namely, use special purpose hardware)
• To analyze large quantities of data
  – Parallel is faster (trades time for money)
  – Relational algorithms exploit parallelism

Moore’s Law

XXX doubles every 18 months
60% increase per year

- Micro-processor speeds
- Chip density
- Magnetic disk density
- Communications bandwidth
  - WAN bandwidth approaching LANs

Implication of Hardware Trends: Clusters

Future Servers are CLUSTERS of processors, discs
Distributed Database techniques make clusters work
Summary cont’d

- Scaleable Networks and Platforms
  - Build clusters of commodity processors & storage
  - Commodity Cluster Operating System is key
  - Fault isolation and tolerance is key
  - Automatic Parallel Programming is key

Outline

- Introduction
- Requirements / performance metrics
  - Parallelism in database systems
    - Partitioning: data, index
    - Split/merge operator
    - Pipelining
    - Operators: aggregates, sorting, join
    - Optimization
- Parallel Database Machines

Summary

- Tech trends => pipeline & partition parallelism
  - Lots of bytes & bandwidth per dollar
  - Lots of latency
  - Lots of MIPS per dollar
  - Lots of processors

The Software Challenge

- Automatic data placement
  (partition: random or organized)
- Automatic parallel programming
  (process placement)
- Parallel concepts, algorithms & tools
- Parallel Query Optimization
- Execution Techniques
  - load balance,
  - checkpoint/restart,
  - multi-programming

Parallelism: Goal=Performance

Goal is to get ‘good’ performance:

Law 1: parallel system should be faster than serial system

Law 2: parallel system should give near-linear scaleup or near-linear speedup or both.

Architecture: Shared What?

Shared Nothing (network)
Shared Disk (SMP)
Shared Memory (SMP)
Architecture: Shared What?

**Shared Nothing (network)**
- Program: Hard Build: Cheap
- Scaleup: Easy
- VMScluster, Sysplex

**Shared Disk**
- Program: Easy Build: Expensive
- Scaleup: Hard
- Tandem, Teradata, SP2

**Shared Memory (SMP)**
- Program: Easy Build: Expensive
- Scaleup: Hard
- CMU SCS

Kinds of Parallel Execution

**Pipeline**
- Any Sequential Program
- Any Sequential Program
- Any Sequential Program

**Partition**
- outputs split N ways
- inputs merge M ways

Parallelism: Speedup & Scaleup

**Speedup:**
- Same Job,
  - More Hardware
  - Less time

**Scaleup:**
- Bigger Job,
  - More Hardware
  - Same time

**Transaction Scaleup:**
- more clients/servers
  - Same response time

CMU SCS

The Perils of Parallelism

**Startup:**
- Creating processes
- Opening files
- Optimization

**Interference:**
- Device (cpu, disc, bus)
  - logical (lock, hotspot, server, log,...)

**Skew:**
- If tasks get very small, variance > service time

**The Good Speedup Curve**
- No Parallelism Benefit
- Linearity

**A Bad Speedup Curve**
- 3-Factors

**A Bad Speedup Curve**
- Speedup and 
  - Processors & Discs

Outline

- Introduction
- Requirements / performance metrics
- **Parallelism in database systems**
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- **Parallel Database Machines**

CMU SCS

CMU SCS

Why are Relational Operators So Successful for Parallelism?
Types of DB parallelism

- What kind of parallelism can we do wrt OPERATORS?
- QUERIES?

Example: Automatic Parallel OR DB

Select image from landsat where date between 1970 and 1990 and overlaps(location, :Rockies) and snow_cover(image) > .7;

Assign one process per processor/disk: find images with right data & location analyze image, if 70% snow, return it

Overview

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Automatic Data Partitioning

Split a SQL table to subset of nodes & disks

How??
Automatic Data Partitioning
Split a SQL table to subset of nodes & disks

Partition within set:
- Range
- Hash
- Round Robin

Queries?

Outline
- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
- Parallel Database Machines

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning

Sensitivity to partitioning:
Shared disk / memory?
Shared nothing?

Good for equi-joins, range queries
group-by

Good for equi-joins

Good to spread load

Good to spread load

Shared disk benefits from "good" partitioning
Index Partitioning

Hash indices partition by hash

B-tree indices partition as a forest of trees.
One tree per range

Primary index clusters data

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Data Rivers: Split + Merge Streams

N X M Data Streams

Producers add records to the river,
Consumers consume records from the river
Purely sequential programming,
River does flow control and buffering
does partition and merge of data records
River = Split/Merge in Gamma =
Exchange operator in Volcano.

Partitioned Execution

Spreads computation and IO among processors

Partitioned data gives NATURAL parallelism

‘Split’ operator

- Eg., for hashing

Picking Data Ranges

Disk Partitioning
For range partitioning, sample load on disks.
Cool hot disks by making range smaller
For hash partitioning,
Cool hot disks
by mapping some buckets to others
Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Blocking Operators=Short Pipelines

An operator is blocking if it does not produce any output until it has consumed all its input.

Examples:
Sort, Aggregates, Hash-Join (reads all of one operand)

Blocking operators kill pipeline parallelism
Bushy trees?

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Simple Aggregates (sort or hash?)

Simple aggregates?

GROUP BY aggregates?

Simple Aggregates (sort or hash?)

Simple aggregates (count, min, max, ...) can use indices
  - More compact
  - Sometimes have aggregate info.

GROUP BY aggregates
  - scan in category order if possible (use indices)
  - Else
    - If categories fit in RAM
      use RAM category hash table
    - Else
      - make temp of <category, item>
      - sort by category
      - do math in merge step.
Parallel Aggregates

For aggregate function, need a decomposition strategy:
\[ \text{count}(S) = \sum \text{count}(s(i)), \text{ditto for sum}() \]
\[ \text{avg}(S) = \sum \text{sum}(s(i)) / \text{count}(s(i)) \]
and so on...

For groups,
sub-aggregate groups close to the source
drop sub-aggregates into a hash river.

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Parallel Sort

Sort is benchmark from hell for shared nothing machines
net traffic = disk bandwidth, no data filtering at the source

M inputs N outputs
Disk and merge
not needed if sort fits in memory

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Hash Join: Combining Two Tables

How parallelize hash join?

Hash Join: Combining Two Tables

Hash smaller table into N buckets (hope N=1)
If N=1 read larger table, hash to smaller
Else, hash outer to disk then bucket-by-bucket hash join.

Purely sequential data behavior
Always beats sort-merge and nested
unless data is clustered.
Good for equi, outer, exclusion join
Lots of papers!

Hash reduces skew
Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

Parallel Hash Join

- ICL implemented hash join with bitmaps in CAFS machine (1976)!
- Kitsuregawa pointed out the parallelism benefits of hash join in early 1980’s (it partitions beautifully)
- Hashing minimizes skew, requires little thinking for redistribution
- Hashing uses massive main memory

Parallel Query Optimization

- Relatively easy to build a parallel executor
- Hard to write a robust optimizer
  - Tricks
  - Complexity barrier
  - Open research!
- Common approach: 2 phases
  - Pick best sequential plan
  - Pick degree of parallelism
  - Bind operators to processors (decorate tree)
- What’s wrong with that?

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
  - Optimization
- Parallel Database Machines

What Systems Work This Way

Shared Nothing
- Teradata: 400 nodes
- Tandem: 80x12 nodes
- IBM / SP2 / DB2: 128 nodes
- Informix/SP2: 100 nodes
- ATT & Sybase: 8x14 nodes

Shared Memory
- Informix: 9 nodes
- RedBrick: ? nodes
Summary

• Why Parallelism:
  – technology push
  – application pull

• Parallel Database Techniques
  – partitioned data
  – partitioned and pipelined execution
  – parallel relational operators

• Optimization still open problem