Parallel Database Systems

Anastassia Ailamaki
http://www.cs.cmu.edu/~natassa

Main Message

- Technology trends give
  - Many processors and storage units
  - Inexpensively

- Parallelism in DBs came from a failed idea
- 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
Implications of Hardware Trends

Large Disc Farms will be inexpensive (10k$/TB)
Large RAM databases will be inexpensive (1k$/GB)
Processors will be inexpensive
So building block will be
a processor
with large RAM
lots of Disc
lots of network bandwidth
CyberBrick™

Implication of Hardware Trends: Clusters

Future Servers are CLUSTERS
of processors, discs
Distributed Database techniques
make clusters work

The Hardware is in Place and
Then A Miracle Occurs

SNAP
Scaleable Network And Platforms
Commodity Distributed OS
built on
Commodity Platforms
Commodity Network Interconnect
Enables
Parallel Applications
Summary

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

- **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

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  Shared Memory (SMP)

Program? Build? Scaleup?

Program? Build? Scaleup?

Program: Easy
Build: Cheap
Scaleup: Easy
Tandem, Teradata, SP2

Program: Hard
Build: Expensive
Scaleup: Hard
Sequent, SGI, Sun

VMScluster, Sysplex
Kinds of Parallel Execution

- **Pipeline**
  - Any Sequential Program

- **Partition**
  - Outputs split N ways
  - Inputs merge M ways

Parallelism: Speedup & Scaleup

- **Speedup:**
  - Same Job, More Hardware
  - Less time
  - Example: 100GB to 100GB

- **Scaleup:**
  - Bigger Job, More Hardware
  - Same time
  - Example: 100GB to 1 TB

Transaction Scaleup:
- More clients/servers
- Same response time

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
Outline

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

Why are Relational Operators So Successful for Parallelism?

Relational data model  uniform operators
on uniform data stream
closed under composition

Each operator consumes 1 or 2 input streams
Each stream is a uniform collection of data
Sequential data in and out: Pure dataflow
partitioning some operators (e.g. aggregates, non-equi-join, sort,...)
requires innovation

AUTOMATIC PARALLELISM

Types of DB parallelism

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

- **Intra-operator**
  - All machines work to execute one operator

- **Inter-operator**
  - Each operator may run concurrently on different sites
  - (exploits pipelining)

- **Inter-query**
  - Different queries run on different sites

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

Outline

- 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

Partition within set:
- Range
- Hash
- Round Robin

Queries?
Automatic Data Partitioning

Split a SQL table to subset of nodes & disks

Partition within set:
- Range
- Hash
- Round Robin

<table>
<thead>
<tr>
<th>Range</th>
<th>Hash</th>
<th>Round Robin</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
<td><img src="https://via.placeholder.com/150" alt="Diagram" /></td>
</tr>
</tbody>
</table>

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

Good for equi-joins, range queries, group-by

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

Outline

- Introduction
- Requirements / performance metrics
- Parallelism in database systems
  - Partitioning: data, index
  - Split/merge operator
  - Pipelining
  - Operators: aggregates, sorting, join
- Parallel Database Machines
Index Partitioning
Hash indices partition by hash
B-tree indices partition as a forest of trees.
   One tree per range
Primary index clusters data

Secondary Index Partitioning
In shared nothing, secondary indices are Problematic. Ideas?
Partition by base table key ranges
   Insert? What about unique?
   Lookup?
   Unique index?

Secondary Index Partitioning
In shared nothing, secondary indices are Problematic
Partition by base table key ranges
   Insert: completely local
   Lookup: examines ALL trees (see figure)
   Unique index involves lookup on insert.
Secondary Index Partitioning

In shared nothing, secondary indices are Problematic

Partition by base table key ranges
Insert: completely local (but what about unique?)
Lookup: examines ALL trees (see figure)
Unique index involves lookup on insert.

Partition by secondary key ranges
Insert?
Lookup?
Unique index?

Teradata solution

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

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

N x M way Parallelism

N inputs, M outputs, no bottlenecks.
Partitioned Data
Partitioned and Pipelined Data Flows
Picking Data Ranges

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

**River Partitioning**
- Use hashing and assume uniform
- If range partitioning, sample data and use histogram to level the bulk

Teradata, Tandem, Oracle use these tricks

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?
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:

\[
\text{count}(S) = \sum \text{count}(s(i)), \text{ditto for sum()}
\]
\[
\text{avg}(S) = \frac{\sum \text{sum}(s(i)))}{\sum \text{count}(s(i))}
\]
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

Sub-sorts generate runs
Merge runs
M inputs N outputs
Disk and merge not needed if sort fits in memory

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

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

---

**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

---

**Exotic Joins (Exclusion, Cartesian, Outer, ...)**

Exclusion used for NOT IN and DIFFERENCE queries

Outer is “lossless” join, (left, right) appears with null sibling if matching sibling not found.

Cartesian is often a mistake (missing where clause) but also important for small-table large-table optimization (also called STAR schema).

Small table used as a pick list.
- Each small table represents a mapping: name -> code
- set membership (e.g. holidays)

Best plan:
- Restrict small tables,
- Form Cartesian product of all small
- Send it to each partition of large table for hash join.
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?

Best parallel plan ≠ best serial plan

Counter-example?
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: 110 nodes
- IBM / SP2 / DB2: 128 nodes
- Informix/SP2: 100 nodes
- ATT & Sybase: 8x14 nodes

Shared Disk
- Oracle: 170 nodes
- Rdb: 24 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