Future of Parallel DBMS

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

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
- Microprocessor speeds
- Chip density
- Magnetic disk density
- Communication bandwidth
  - WAN approaches LAN

1 chip memory size
- 2 MB to 32 MB

Implication of Hardware Trends:
Clusters

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

Implications

- Tech trends => pipeline & partition parallelism
- Lots of bytes & bandwidth per dollar
- Lots of latency
- Lots of MIPS per dollar
- Lots of processors
Implications 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 (hard!)

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
  - How to partition: randomly or organized
- Automatic parallel programming
  - Essentially: process placement
- Parallel concepts, algorithms & tools
- Parallel Query Optimization
- Execution Techniques
  - load balancing
  - checkpoint/restart
  - multi-programming
Parallelism: Goal=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?

<table>
<thead>
<tr>
<th>Shared Nothing (network)</th>
<th>Shared Disk</th>
<th>Shared Memory (SMP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program?</td>
<td>Program?</td>
<td>Program?</td>
</tr>
<tr>
<td>Build?</td>
<td>Build?</td>
<td>Build?</td>
</tr>
<tr>
<td>Scaleup?</td>
<td>Scaleup?</td>
<td>Scaleup?</td>
</tr>
</tbody>
</table>

| Sequent, SGI, Sun       | VMScluster, Sysplex |
| Sequent, Sun            | Tandem, Teradata, SP2 |
Kinds of Parallel Execution

Pipeline
- 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

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. aggregation, non-equi-join, sort,..)
  - requires innovation
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

```sql
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

How??

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

Queries?

Sensitivity to partitioning:
- Shared disk / memory?
- Shared nothing?
Automatic Data Partitioning

Split a SQL table to subset of nodes & disks

Partition within set:
- **Range**
- **Hash**
- **Round Robin**

- Good for equi-joins, range queries, group-by
- Good for equi-joins
- Good to spread load

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
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
‘Split’ operator

- E.g., 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:
  count(S) = Σ count(a(i)), ditto for sum()
  avg(S) = (Σ sum(a(i))) / Σ count(a(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

- River is range or hash partitioned
- 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
Hash Join: Combining Two Tables

How parallelize hash join?

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
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 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?

Parallel Query Optimization

- Best parallel plan != best serial plan
- 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 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