Parallel Databases

CS227, Spring 2011

Yang Lu
James Tavares
Overview

- Motivations
- Architectures
- Partitioning Schemes
- Relational Operator Parallelism
  - Parallel Sort, Join, Selection, etc.
- Gamma
  - Architecture, Performance Analysis
- XPRS Design
Why parallel database?

• Driving force
  – Demand on storing and analyzing large volumes of data
  – Demand on high throughput for transaction processing

• Prices of microprocessors, memory and disks have dropped sharply

• Relational databases are ideally suited to parallelization.
Relation database parallelization

- Relations can be partitioned on multiple disks.
  - Horizontal partitioning: tuples of a relation are divided among many disks.
- Operations can be executed in parallel
  - Pipelined parallelism

Diagram:

- Pipeline parallelism
- Partitioned data allows partitioned parallelism

Operations can be executed in parallel
- Pipelined parallelism
Interconnection Networks

(a) bus

(b) mesh

(c) hypercube
Architectures

• shared-memory:
  – share direct access to a common global.

• shared-disks
  – Each processor has direct access to all disks.

• shared-nothing:
  – The Teradata, Tandem, Gamma
Architectures

- Shared Memory Multiprocessor
- Shared Disk Multiprocessor
Partitioning a Relation across Disks

• Principles
  – It is better to assign a small relation to a single disk.
  – Large relations are preferably partitioned across all the available disks
    • $m$ disk blocks and $n$ disks
    • should be allocated $\min(m,n)$ disks

• Techniques
  – Round-robin
  – Hash partitioning
  – Range partitioning
Partitioning Techniques

**Round-robin:**
Send the $i^{th}$ tuple inserted in the relation to disk $i \mod n$.

**Hash partitioning:**

- Choose one or more attributes as the partitioning attributes.
- Choose hash function $h$ with range $0...n - 1$
- Let $i$ denote result of hash function $h$ applied to the partitioning attribute value of a tuple. Send tuple to disk $i$. 
Partitioning Techniques

- **Range partitioning:**
  - Choose an attribute as the partitioning attribute.
  - A partitioning vector \([v_0, v_1, ..., v_{n-2}]\) is chosen.
  - Let \(v\) be the partitioning attribute value of a tuple. Tuples such that \(v_i \leq v_{i+1}\) go to disk \(i+1\). Tuples with \(v < v_0\) go to disk 0 and tuples with \(v \geq v_{n-2}\) go to the last disk.
Comparison of Partitioning Techniques

- A. Sequential scan
- B. Point queries.
  E.g. employee-name=“Campbell”.
- C. Range queries.
  E.g. 10000<salary<20000
Parallelism Hierarchy

• Interquery
  – Queries/transactions execute in parallel with one another
    • Locking and logging must be coordinated by passing messages between processors.
    • Cache-coherency has to be maintained

• Intraquery
  – Execution of a single query in parallel on multiple processors
Parallelism Hierarchy

• Two complementary forms of intraquery parallelism:

  – **Intraoperation Parallelism** – parallelize the execution of each individual operation in the query.

  – **Interoperation Parallelism** – execute the different operations in a query expression in parallel.
Parallel Sort

• Range-Partitioning Sort
  – Redistribution using a range-partition strategy
  – Each processor sorts its partition locally

• Parallel External Merge-Sort
  – Each processor $P_i$ locally sorts the data on disk $D_i$.
  – The sorted runs on each processor are then merged.
Parallel Join

• Partitioned Join
  – Use the same partitioning function on both relations
    • Range partitioning on the join attributes
    • Hash partitioning on the join attributes
  – Equi-joins and natural joins
Partitioned Join

\[ r_0 \rightarrow P_0 \rightarrow s_0 \]

\[ r_1 \rightarrow P_1 \rightarrow s_1 \]

\[ r_2 \rightarrow P_2 \rightarrow s_2 \]

\[ r_3 \rightarrow P_3 \rightarrow s_3 \]

\[ r \rightarrow \ldots \rightarrow s \]
Partitioned Parallel Hash-Join

• Simple Hash-Join
  – Route tuples to their appropriate joining site.
  – The smaller joining relation staged in an in-memory hash (which is formed by hashing on the join attribute of each tuple).
  – Tuples of the larger joining relations probe the hash table for matches.

• Other optimization: Hybrid Hash-Join
Parallel Join

- Fragment-and-Replicate Join
  - Partitioning not possible for some join conditions
    - E.g., non-equijoin conditions, such as $r.A > s.B$.

- **fragment and replicate** technique
Fragment-and-Replicate Join

(a) Asymmetric fragment and replicate

(b) Fragment and replicate
Interoperator Parallelism

• Pipelined Parallelism
  – The output tuples of one operation are consumed by a second operation.
  – No need to write any of the intermediate results to disk.
Pipelined parallelism

– Consider a join of four relations

\[ r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 \]

• Let P1 be assigned the computation of \( \text{temp1} = r_1 \bowtie r_2 \)
• Let P2 be assigned the computation of \( \text{temp2} = \text{temp1} \bowtie r_3 \)
• And P3 be assigned the computation of \( \text{temp2} \bowtie r_4 \)
Measuring DB Performance

• Throughput
  – The number of tasks, or the size of task, that can be completed in a given time interval

• Response Time
  – The amount of time it takes to complete a single task from the time it is submitted

• Goal: improve both through parallelization
Absolute vs. Relativistic

• Absolute
  – Q: Does system meet my requirements?
  – Q: How does system compare with system Y?

• Relativistic
  – As some resource is varied, determine how system scales and how speed is affected
  – Q: Will increased resources let me process larger datasets?
  – Q: Can I speed up response time by adding resources?
Scaleup

- Baseline: Task Q runs on $M_S$ in $T_S$ seconds
- Task $Q_N$ runs on $M_L$ in $T_L$ seconds
- $Q_N, M_L$ are $N$ times larger than $Q, M_S$, respectively
- Scaleup $= \frac{T_S}{T_L}$
  - Linear: $T_S = T_L$
  - Sublinear: $T_L > T_S$
Speedup

- Task Q runs on $M_S$ and responds in time $T_S$.
- Same task Q runs on $M_L$ and responds in time $T_L$.
  - Goal: $T_L$ should be time: $T_S \times (S/L)$.
- Speedup = $T_S / T_L$.

Diagram:
- Blue line: linear speedup.
- Red line: sublinear speedup.

Speedup graph with $T$ on the y-axis and resources on the x-axis.
Performance Factors

• Interference
  – Parallel processes compete for shared resources (e.g., system bus, network, or locks)

• Start-up costs
  – Associated with initiating a single process
  – Start-up time may overshadow processing time

• Skew
  – Difficult to subdivide tasks into equal-sized parts
  – Most-skewed subtask governs response time
Gamma Overview

• First operational prototype 1985, U. of Wisconsin
• Shared-nothing architecture
  – Interconnected by communications network
  – Promotes commodity-based hardware, lots of processors
• Hash-based parallel algorithms to disburse load
Gamma Hardware

• Version 1.0
  – (18) VAX 11/750 machines, with 2MB RAM
  – 8 machines with 333 MB HD; balance is diskless
  – 80mbit/s token ring, 4mbit/s at each CPU

• Version 2.0
  – 32x Intel 386 iPSC/2 hypercube CPUs, with 8MB RAM
  – 330 MB HDD per CPU
  – 8 x 22.4Mbps/s serial hypercube channels
Gamma Storage Engine

• Horizontally Partitioned
  – Round robin, hashed, or range partitioned
  – For performance analysis:
    • Hashed for source relations
    • Round-robin for destination relations

• Clustered and non-clustered indexes offered within each partition
  – Clustered index allowed on non-partition attribute
Recovery: Chained Declustering

- Assume N nodes, and N fragments of R, \( R_N \)
- Backup copy stored at node: \((i+1) \mod N\)
- On failure, nodes assumes \(1/(N-1)\) of the load
- Multiple failures permitted as long as no two adjacent nodes fail together

<table>
<thead>
<tr>
<th>Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Copy</td>
<td>R0</td>
<td>---</td>
<td>( \frac{1}{7} )R2</td>
<td>( \frac{2}{7} )R3</td>
<td>( \frac{3}{7} )R4</td>
<td>( \frac{4}{7} )R5</td>
<td>( \frac{5}{7} )R6</td>
<td>( \frac{6}{7} )R7</td>
</tr>
<tr>
<td>Backup Copy</td>
<td>( \frac{1}{7} )r7</td>
<td>---</td>
<td>r1</td>
<td>( \frac{6}{7} )r2</td>
<td>( \frac{5}{7} )r3</td>
<td>( \frac{4}{7} )r4</td>
<td>( \frac{3}{7} )r5</td>
<td>( \frac{2}{7} )r6</td>
</tr>
</tbody>
</table>
Gamma Architecture

- One per active user
- One per active query
- One per database
- >=1 per active tree node
Gamma Operator & Split Table

Operators Include:
SCAN, SELECT, JOIN, STORE, UPDATE, etc.

<table>
<thead>
<tr>
<th>Value</th>
<th>Destination Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Processor #3, Port #5)</td>
</tr>
<tr>
<td>1</td>
<td>(Processor #2, Port #13)</td>
</tr>
<tr>
<td>2</td>
<td>(Processor #7, Port #6)</td>
</tr>
<tr>
<td>3</td>
<td>(Processor #9, Port #15)</td>
</tr>
</tbody>
</table>
Example Query

Step 1: Query Parsed, Optimized, Compiled

Step 2: Scheduler Process Assigned by Query Manager

Partitions A,B to Nodes 1,2

Step 3: A.SELECT and B.SCAN processes started on Nodes 3,4

Step 4: Scheduler start JOIN processes on Nodes 1,2

Step 5: Split Table Partitions A,B to Nodes 1,2

Step 6: Partitioned Hash Join using Nodes 1,2

Step 7: JOIN results round-robin to Nodes 3,4

Step 8: Scheduler completes, Query Manager returns result
Nonindexed Selections (seconds)

Gamma Loses 1%, Wins 10%

Gamma 1%
Teradata 1%
Gamma 10%
Teradata 10%

0 20 40 60 80 100 120 140 160
10,000 tuples 100,000 tuples 1,000,000 tuples

Gamma 1%  Teradata 1%  Gamma 10%  Teradata 10%
Non-clustered Indexed Selections (seconds)

Gamma’s B+ Tree outperforms Teradata’s unordered indexes

- Gamma 1%
- Teradata 1%
Selection Speedup

Nonindexed Selection

Indexed Selection

I/O Bound

Network Bound

Overhead
Gamma Join Performance

• Relations
  – A – 100,000 tuples
  – Bprime – 10,000 tuples
  – A \bowtie Bprime – 10,000 tuples

• Join Types
  – Local
    • join occurs only on disk nodes
  – Remote
    • join occurs only on disk-less nodes
  – Allnodes
    • join occurs on both disk and disk-less nodes
  – Scans always run on respective disk node
Join $A, B'$ Speedup

**Join Attr = Partitioning Attr**

- Local Joins
- Remote Joins
- Allnodes Joins

**Join Attr $\neq$ Partitioning Attr**

- Local Joins
- Remote Joins
- Allnodes Joins

**System:**
- Constant Memory (no overflow)
- 4 Kbyte disk pages
- 100,000 tuple relations

**Query:**
- $\text{JoinABprime}$
  - (Join attrs = Partitioning Attrs)
Join A, Bprime Response Time

**Join Attr = Partitioning Attr**

- RESPONSE TIME (SECONDS)

- System:
  - Constant Memory (no overflow)
  - 4 Kbyte disk pages
  - 100,000 tuple relations

- Query:
  - JoinABprime (Join attrs = Partitioning Attrs)

- Local Joins
- Remote Joins
- Allnodes Joins

**Join Attr != Partitioning Attr**

- RESPONSE TIME (SECONDS)

- System:
  - Constant Memory (no overflow)
  - 4 Kbyte disk pages
  - 100,000 tuple relations

- Query:
  - JoinABprime (Join attrs <> Partitioning Attrs)

- Local Joins
- Remote Joins
- Allnodes Joins

**Local Wins**

**Remote Wins**
Gamma Join Overflow Performance

- Simple Hash Join w/ Join Attr. = Part. Attr
- Memory was incrementally reduced
- Performance crossover
- Why? Overflows handled by recursive joins
  - With new hash function!
  - New hash equiv. of:
    Join Attr. != Part. Attr
Gamma (V2) Scaleup – Join A,Bprime

- **Intel Hypercube**
- **Ranges**
  - CPUs: [5, 30]
  - “A” relation: [1M, 6M]
  - “Bprime” relation: [100k, 600k]
- **Factors**
  - Scheduler on single CPU
  - Diminished short-circuiting
  - Communications network

![Response Time Graph](image)
XPRS Overview

• Proposed extension to POSTGRES
  – 2-D file allocation, and RAID
  – Parallel queries, fast path, partial indexes
  – Special purpose concurrency
  – Parallel query plans

• Architecture
  – Shared-memory (faster than network)
  – General-purpose OS
  – Large Sun machine or a SEQENT Symmetry
2-D File System

- A file is defined by:
  - Starting disk
  - Width, in disks
  - Height, in tracks
  - Starting track on each disk

- Larger Widths
  - Increase throughput
  - Minimize “hot spots”

- Each “Logical Disk” is a group of physical disks, protected by RAID5
Changes to POSTQUEL

- **Parallel** keyword alerts DBMS to statements that can be executed in parallel (inter-query parallelism)
  - RETRIEVE... PARALLEL RETRIEVE... PARALLEL RETRIEVE...

- **Fast Path**
  - Allow users to define stored procedures, which run pre-compiled plans with given arguments
  - Bypass: Type checking, parsing, and query optimization

- **Partial Indexes**
  - E.g.: INDEX on EMP(salary) WHERE age < 20
  - Reduces index size, increases performance

- **Range-partitioned Relations**
  - E.g.: EMP where age < 20 TO file1
  - E.g.: EMP where age >= 20 TO file2
Special Purpose Concurrency

• Exploit transactions that *failure commute*

• E.g.: Given two bank withdrawals
  – Both will succeed if there are sufficient funds
  – The failure of one has no impact on the other

• Idea: Mark transaction in class “C1” or “C2”
  – Allow C1 transactions to run concurrently with each other, but not with C2 transactions
  – E.g.: Withdrawal as C1, Transfer as C2
Parallel Query Planner

- Find \( \text{BIG} = \min(\text{RAM}_\text{Needed}, \text{Total}_\text{RAM}) \)
- Find *optimal* sequential plan for memory intervals:
  - \([\text{BIG}, \text{BIG}/2], [\text{BIG}/2, \text{BIG}/4], \ldots, [\text{BIG}/n, 0]\)
- Explore all possible parallel plans of each sequential plan
  - With a sprinkle of heuristics to limit plan space
- Use *optimal* parallel plan
Conclusions

• Parallel DBs important to meet future demands
• Historical context important
• Proved many can be made to perform the work of one, only better
• Horizontal partitioning effective
• Speedup and scaleup is possible, at least for sufficiently “small” node counts
Questions?