CompSci 516
Database Systems

Lecture 24
Parallel DBMS

Instructor: Sudeepa Roy
Review (Selinger Query Opt)
Where are we now?

Relational model, queries, db design
- Relational Model
- Normal Forms, FD
- Query in SQL / RA / RC
- Recursion

Beyond Relational Model
- XML
- NOSQL
- JSON/MongoDB

DBMS Internals and Query Processing
- Storage
- Index
- Join algo/Sorting
- Execution/Optimization

Transactions
- Basics
- Concurrency Control
- Recovery

(Basic) Big Data Processing
- Map-Reduce/Spark
- Parallel DBMS
- Distributed DBMS

Other Topics
- Data Mining
- Data Cube

Covered

To be covered

Duke CS, Spring 2022
CompSci 516: Database Systems
Announcements (Tues, 4/5)

• HW3 due today noon

• Focus on Project from now on - no more graded quiz!
  – Due 4/13 next Wed + extra time until Friday noon
  – For everything -- report, code, video
  – Do not focus on GUI – anything basic and functional is fine

• 5 mins presentation video – instructions to be posted on Ed

• Course evaluations open – your feedback is very important for improving the class
  – Small token of appreciation 75% - 2 extra points and 90% - 4 extra points to your final exam score for everyone in class
Reading Material

• [RG]
  – Parallel DBMS: Chapter 22.1-22.5
  – Distributed DBMS: Chapter 22.6 – 22.14

• [GUW]
  – Parallel DBMS and map-reduce: Chapter 20.1-20.2
  – Distributed DBMS: Chapter 20.3, 20.4.1-20.4.2, 20.5-20.6

• Other recommended readings:
  – Chapter 2 (Sections 1,2,3) of Mining of Massive Datasets, by Rajaraman and Ullman: http://i.stanford.edu/~ullman/mmds.html
  – Original Google MR paper by Jeff Dean and Sanjay Ghemawat, OSDI’ 04: http://research.google.com/archive/mapreduce.html

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
Parallel DBMS
# Parallel vs. Distributed DBMS

<table>
<thead>
<tr>
<th>Parallel DBMS</th>
<th>Distributed DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Parallelization of various operations</td>
<td>• Data is physically stored across different sites</td>
</tr>
<tr>
<td>– e.g., loading data, building indexes, evaluating queries</td>
<td>– Each site is typically managed by an independent DBMS</td>
</tr>
<tr>
<td>• Data may or may not be distributed initially</td>
<td>• Location of data and autonomy of sites have an impact on Query opt., Conc. Control and recovery</td>
</tr>
<tr>
<td>• Distribution is governed by performance consideration</td>
<td>• Also governed by other factors:</td>
</tr>
<tr>
<td></td>
<td>– increased availability for system crash</td>
</tr>
<tr>
<td></td>
<td>– local ownership and access</td>
</tr>
</tbody>
</table>
Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

1,000 x parallel
1.5 minute to scan.

Parallelism:
divide a big problem
into many smaller ones
to be solved in parallel.
Parallel DBMS

• Parallelism is natural to DBMS processing
  – Pipeline parallelism: many machines each doing one step in a multi-step process.
  – Data-partitioned parallelism: many machines doing the same thing to different pieces of data.
  – Both are natural in DBMS!

Pipeline

Partition

outputs split N ways, inputs merge M ways
DBMS: The parallel Success Story

• DBMSs are the most successful application of parallelism
  – Teradata (1979), Tandem (1974, later acquired by HP),.. 
  – Every major DBMS vendor has some parallel server

• Reasons for success:
  – Bulk-processing (= partition parallelism)
  – Natural pipelining
  – Inexpensive hardware can do the trick
  – Users/app-programmers don’t need to think in parallel
Some Terminology

- **Speed-Up**
  - More resources means proportionally less time for given amount of data.

- **Scale-Up**
  - If resources increased in proportion to increase in data size, time is constant.

---

Ideal graphs

- **Ideal:** linear speed-up

- **Ideal:** linear scale-up

---

Duke CS, Spring 2022
CompSci 516: Database Systems
Some Terminology

In practice

• Due to overhead in parallel processing

• Start-up cost
Starting the operation on many processors, might need to distribute data

• Interference
Different processors may compete for the same resources

• Skew
The slowest processor (e.g. with a huge fraction of data) may become the bottleneck

Ideal: linear speed-up

Actual: sub-linear speed-up

Ideal: linear scale-up

Actual: sub-linear scale-up

#CPUs (degree of ||-ism)

#ops/sec
(throughput)

#CPUs + size of database
degree of ||-ism

Duke CS, Spring 2022

CompSci 516: Database Systems
Basics of Parallelism

• Units: a collection of processors
  – assume always have local cache
  – may or may not have local memory or disk (next)

• A communication facility to pass information among processors
  – a shared bus or a switch

• Different architecture
  – Whether memory AND/OR disk are shared
Shared Memory

- Easy to program
- Expensive to build
- Low communication overhead: shared mem.
- Difficult to scaleup (memory contention)

Interconnection Network

Global Shared Memory

Duke CS, Spring 2022
Shared Disk

- Trade-off but still interference like shared-memory (contention of memory and nw bandwidth)

Diagram:
- Local memory
- Interconnection Network
- Shared disk
Shared Nothing

- Hard to program and design parallel algs
- Cheap to build
- Easy to scaleup and speedup

Considered to be the best architecture

We will assume this architecture!

Interconnection Network

- local memory and disk
- no two CPU can access the same storage area
- all communication through a network connection

- • Hard to program and design parallel algs
  • Cheap to build
  • Easy to scaleup and speedup

Considered to be the best architecture

We will assume this architecture!
Different Types of DBMS Parallelism

• Intra-operator parallelism
  – get all machines working to compute a given operation (scan, sort, join)
  – OLAP (decision support)

• Inter-operator parallelism
  – each operator may run concurrently on a different site (exploits pipelining)
  – For both OLAP and OLTP

• Inter-query parallelism
  – different queries run on different sites
  – For OLTP

• We’ll focus on intra-operator parallelism
Data Partitioning

Horizontally Partitioning a table (why horizontal?):

Range-partition

- Good for equijoins, range queries, group-by
- Can lead to data skew

Hash-partition

- Good for equijoins
- But only if hashed on that attribute
- Can lead to data skew

Block-partition or Round Robin

- Send i-th tuple to i-mod-n processor
- Good to spread load
- Good when the entire relation is accessed

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning
Best serial plan may not be best

• Why?
• Trivial counter-example:
  – Table partitioned with local secondary index at two nodes
  – Range query: all of node 1 and 1% of node 2.
  – Node 1 should do a scan of its partition.
  – Node 2 should use secondary index.
Example problem: Parallel DBMS

R(a,b) is horizontally partitioned across N = 3 machines.

Each machine locally stores approximately 1/N of the tuples in R.

The tuples are randomly organized across machines (i.e., R is block partitioned across machines).

Show a RA plan for this query and how it will be executed across the N = 3 machines.
Pick an efficient plan that leverages the parallelism as much as possible.

- SELECT a, max(b) as topb
- FROM R
- WHERE a > 0
- GROUP BY a
R(a, b)

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
If more than one relation on a machine, then “scan S”, “scan R” etc

```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```
SELECT a, \text{max}(b) \text{ as topb} \\
\text{FROM R} \\
\text{WHERE} a > 0 \\
\text{GROUP BY} a
\[ \gamma_{a, \text{max}(b) \rightarrow b} \]

\[ \sigma_{a > 0} \]

\[ \text{scan} \]

Machine 1

1/3 of R

\[ \gamma_{a, \text{max}(b) \rightarrow b} \]

\[ \sigma_{a > 0} \]

\[ \text{scan} \]

Machine 2

1/3 of R

\[ \gamma_{a, \text{max}(b) \rightarrow b} \]

\[ \sigma_{a > 0} \]

\[ \text{scan} \]

Machine 3

1/3 of R

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

R(a, b)
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

\[ R(a, b) \]

Hash on a
\[ \gamma_{a, \text{max}(b)} \rightarrow b \]
\[ \sigma_{a > 0} \]
scan
Machine 1
1/3 of R

Hash on a
\[ \gamma_{a, \text{max}(b)} \rightarrow b \]
\[ \sigma_{a > 0} \]
scan
Machine 2
1/3 of R

Hash on a
\[ \gamma_{a, \text{max}(b)} \rightarrow b \]
\[ \sigma_{a > 0} \]
scan
Machine 3
1/3 of R
SELECT a, max(b) as topb FROM R WHERE a > 0 GROUP BY a

R(a, b)

Hash on a

γa, max(b)→ b

σa>0

scan

Machine 1

1/3 of R
SELECT a, max(b) as topb FROM R WHERE a > 0 GROUP BY a
Benefit of hash-partitioning

• What would change if we hash-partitioned R on R.a before executing the same query on the previous parallel DBMS

SELECT a, max(b) as topb FROM R WHERE a > 0 GROUP BY a
SELECT a, max(b) as topb FROM R WHERE a > 0 GROUP BY a
Hash-partition on a for R(a, b)

- It would avoid the data re-shuffling phase
- It would compute the aggregates locally

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
SELECT a, max(b) as topb FROM R WHERE a > 0 GROUP BY a
Column Store (overview)
Row vs. Column Store

• **Row store**
  – store all attributes of a tuple together
  – storage like “row-major order” in a matrix

• **Column store**
  – store all rows for an attribute (column) together
  – storage like “column-major order” in a matrix

• **e.g.**
  – MonetDB, Vertica (earlier, C-store), SAP/Sybase IQ, Google Bigtable (with column groups)
What is a column-store?

**row-store**
- Date
- Store
- Product
- Customer
- Price

- + easy to add/modify a record
- - might read in unnecessary data

**column-store**
- Date
- Store
- Product
- Customer
- Price

- + only need to read in relevant data
- - tuple writes require multiple accesses

=> suitable for read-mostly, read-intensive, large data repositories
Telco Data Warehousing example

1 Typical DW installation

1 Real-world example

“One Size Fits All? - Part 2: Benchmarking Results” Stonebraker et al. CIDR 2007

QUERY 2
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in ("AE", "AA")
AND usage.toll_price > 0
AND source.type != "CIBER"
AND toll.rating_method = "IS"
AND usage.invoice_date = 20051013
GROUP BY account.account_number

<table>
<thead>
<tr>
<th></th>
<th>Column-store</th>
<th>Row-store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query 1</td>
<td>2.06</td>
<td>300</td>
</tr>
<tr>
<td>Query 2</td>
<td>2.20</td>
<td>300</td>
</tr>
<tr>
<td>Query 3</td>
<td>0.09</td>
<td>300</td>
</tr>
<tr>
<td>Query 4</td>
<td>5.24</td>
<td>300</td>
</tr>
<tr>
<td>Query 5</td>
<td>2.88</td>
<td>300</td>
</tr>
</tbody>
</table>

Why? Three main factors (next slides)

Ack: Slide from VLDB 2009 tutorial on Column store
Telco example explained (1/3): read efficiency

row store

read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

column store

read only columns needed

in this example: 7 columns

caveats:
- "select * " not any faster
- clever disk prefetching
- clever tuple reconstruction

Ack: Slide from VLDB 2009 tutorial on Column store
Telco example explained (2/3): compression efficiency

1 Columns compress better than rows
   1 Typical row-store compression ratio 1 : 3
   1 Column-store 1 : 10

1 Why?
   1 Rows contain values from different domains
      \rightarrow more entropy, difficult to dense-pack
   1 Columns exhibit significantly less entropy

1 Caveat: CPU cost (use lightweight compression)
Telco example explained (3/3): sorting & Indexing efficiency

1. Compression and dense-packing free up space
   1. Use multiple overlapping column collections
   1. Sorted columns compress better
   1. Range queries are faster
   1. Use sparse clustered indexes