ADMINISTRIVIA

• OQ5 Due Tonight (11:00)
• HW6 Due next Wednesday (Feb 28)
• HW4 Grades Out
• Cost-estimation problems
  • Fair game for final exam
  • Look at previous midterms early
  • Also in last quarter’s final
“DO YOU KNOW WHAT'S HAPPENING IN 344?”

idkSQL++DB

→∃x
WHAT IS HAPPENING?

What we have been doing

• How to interface with data (SQL, Datalog, SQL++)
• Query execution on a single node (RA, cost estimation)

Next few lectures

• Parallel query execution across multiple nodes

Why do we care about how the query executes?

• Identify where to speed up (indexes)
• Identify where to eliminate bottlenecks
Parallel architectures and querying options available

- Shared memory, shared disk, or shared nothing
- Inter-query, inter-operator, intra-operator

Execution on a shared-nothing, intra-operator model

- Data partitioning in distributed systems
- Grouping execution
- Join execution
WHY COMPUTE IN PARALLEL?

Access to multiple cores

• Most processors have multiple cores
• This trend will likely increase in the future

“Big Data” size issue

• Too large to fit in main memory
• Too large to fit on a single disk
• Distributed query processing on 100x-1000x servers
• Accessible via cloud services (Azure, AWS, ...)

PERFORMANCE METRICS FOR PARALLEL DBMS

Nodes = processors, computers

**Speedup:**
- More nodes, same data $\rightarrow$ higher speed

**Scaleup:**
- More nodes, more data $\rightarrow$ same speed
LINEAR V.S. NON-LINEAR SPEEDUP

![Graph showing linear vs. non-linear speedup with speedup on the y-axis and number of nodes on the x-axis. The graph includes a line labeled "Ideal." At x1, ×5, ×10, and ×15 nodes, the speedup is compared to the ideal speedup.]
LINEAR V.S. NON-LINEAR SCALEUP

Scaleup

# nodes AND data size

×1  ×5  ×10  ×15
WHY SUB-LINEAR SPEEDUP AND SCALEUP?

Overhead → Startup cost

• Cost of starting an operation on many nodes

Interference

• Contention for resources between nodes
• Waiting for other nodes to finish

Data distribution → Skew

• Slowest node becomes the bottleneck
ARCHITECTURES FOR PARALLEL DATABASES

Solutions:

- Shared memory
- Shared disk
- Shared nothing
SHARED MEMORY

Nodes share both RAM and disk
Dozens to hundreds of processors

Example: Azure SQL Server
Check out HW3 query plans (SSMS/Datagrip)

Easy to use and program
Expensive to scale
SHARED DISK

All nodes access the same disks
Found in the largest "single-box" (non-cluster) multiprocessors

Example: Oracle

No need to worry about shared memory

Still hard to scale

Existing deployments typically have fewer than 10 machines
Cluster of commodity machines on high-speed network
Each machine has its own memory and disk: lowest contention.

Examples: Amazon EC2, Google Compute Engine

Easy to maintain and scale
Most difficult to administer and tune.
APPROACHES TO PARALLEL QUERY EVALUATION

Inter-query parallelism
• Transaction per node
• Good for transactional workloads

Inter-operator parallelism
• Operator per node
• Good for analytical workloads

Intra-operator parallelism
• Operator on multiple nodes
• Good for both?
DISTRIBUTED QUERY PROCESSING

Data is horizontally partitioned on many servers

Operators may require data reshuffling

First let’s discuss how to distribute data across multiple nodes / servers
HORIZONTAL DATA PARTITIONING

Data: K A B ...

Servers: 1 2 ... P
HORIZONTAL DATA PARTITIONING

Data:

Servers:

Which tuples go to what server?
HORIZONTAL DATA PARTITIONING

**Block Partition:**
- Partition tuples arbitrarily s.t. size(R₁) \( \approx \ldots \approx \) size(Rₚ)

**Hash partitioned on attribute A:**
- Tuple t goes to chunk i, where i = h(t.A) mod P + 1
- Recall: calling hash fn is free in this class

**Range partitioned on attribute A:**
- Partition the range of A into \(-\infty = v₀ < v₁ < \ldots < vₚ = \infty\)
- Tuple t goes to chunk i, if \(v_{i-1} < t.A < vᵢ\)
UNIFORM DATA V.S. SKEWED DATA

Let \( R(K,A,B,C) \); which of the following partition methods may result in skewed partitions?

**Block partition**
- Uniform

**Hash-partition**
- On the key \( K \)
- On the attribute \( A \)
- Assuming good hash function
  - Uniform
- May be skewed
  - E.g. when all records have the same value of the attribute \( A \), then all records end up in the same partition

Keep this in mind in the next few slides
PARALLEL EXECUTION OF RA OPERATORS: GROUPING

Data: \( R(K,A,B,C) \)

Query: \( \gamma_{A,\text{sum}(C)}(R) \)

How to compute group by if:

R is hash-partitioned on A ?

R is block-partitioned ?

R is hash-partitioned on K ?
PARALLEL EXECUTION OF RA OPERATORS: GROUPING

Data: $R(K,A,B,C)$

Query: $\gamma_{A, \text{sum}(C)}(R)$

$R$ is block-partitioned or hash-partitioned on $K$

Reshuffle $R$ on attribute $A$

Run grouping on reshuffled partitions
SPEEDUP AND SCALEUP

Consider:

- Query: $\gamma_{A,\text{sum}(C)}(R)$
- Runtime: only consider I/O costs

If we double the number of nodes $P$, what is the new running time?

- Half (each server holds $\frac{1}{2}$ as many chunks)

If we double both $P$ and the size of $R$, what is the new running time?

- Same (each server holds the same # of chunks)

But only if the data is without skew!
Informally: we say that the data is skewed if one server holds much more data than the average

E.g. we hash-partition on A, and some value of A occurs very many times (“Justin Bieber”)

Then the server holding that value will be skewed
PARALLEL EXECUTION OF RA OPERATORS: PARTITIONED HASH-JOIN

Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) \Join S(K2, B, C)

• Initially, both R and S are partitioned on K1 and K2

Reshuffle R on R.B and S on S.B

Each server computes the join locally
**Data:** \( R(K_1, A, B), S(K_2, B, C) \)

**Query:** \( R(K_1, A, B) \bowtie S(K_2, B, C) \)

**PARALLEL JOIN ILLUSTRATION**

<table>
<thead>
<tr>
<th>R1</th>
<th>S1</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1, B</td>
<td>K2, B</td>
</tr>
<tr>
<td>1, 20</td>
<td>101, 50</td>
</tr>
<tr>
<td>2, 50</td>
<td>102, 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R2</th>
<th>S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1, B</td>
<td>K2, B</td>
</tr>
<tr>
<td>3, 20</td>
<td>201, 20</td>
</tr>
<tr>
<td>4, 20</td>
<td>202, 50</td>
</tr>
</tbody>
</table>

**Partition**

**Shuffle on B**

**Local Join**
Data: R(A, B), S(C, D)
Query: R(A,B) \bowtie_{B=C} S(C,D)

BROADCAST JOIN

Reshuffle R on R.B
Broadcast S

Why would you want to do this?
Find all orders from today, along with the items ordered

**EXAMPLE PARALLEL QUERY PLAN**

```
SELECT * 
FROM Order AS O, Item AS i 
WHERE O.item = i.item 
  AND O.date = today()
```

Order(oid, item, date), Item(item, …)
Order(oid, item, date), Line(item, ...)

PARALLEL QUERY PLAN

Node 1

hash

h(O.item)

select

O.date = today()

scan

Order O

Node 2

hash

h(O.item)

select

O.date = today()

scan

Order O

Node 3

hash

h(O.item)

select

O.date = today()

scan

Order O

Node 1

Node 2

Node 3
PARALLEL QUERY PLAN

Order(oid, item, date), Line(item, ...)

Node 1

Node 2

Node 3

scan Item i

hash h(i.item)

join o.item = i.item

date = today()

Order o

Item i

scan Item i

hash h(i.item)

hash h(i.item)

scan Item i

Node 1

Node 2

Node 3
Contains all orders and all lines where hash(item) mod 3 + 1 = 1

Contains all orders and all lines where hash(item) mod 3 + 1 = 2

Contains all orders and all lines where hash(item) mod 3 + 1 = 3

Order(oid, item, date), Line(item, ...)

Parallel Query Plan

Node 1

Join

o.item = i.item

Node 2

Join

o.item = i.item

Node 3

Join

o.item = i.item
A CHALLENGE

Have P number of servers (e.g. P=1000)

How do we compute this Datalog query in one step?

Q(x,y,z) :- R(x,y), S(y,z), T(z,x)
HYPERCUBE JOIN

Have P number of servers (e.g. P=1000)

How do we compute this Datalog query **in one step?**

\[ Q(x,y,z) = R(x,y), S(y,z), T(z,x) \]

Organize the P servers into a cube with side \( P^{\frac{1}{3}} \)

• Thus, each server is uniquely identified by \((i,j,k), i,j,k \leq P^{\frac{1}{3}}\)

**Step 1:**

• Each server sends \( R(x,y) \) to all servers \((h(x),h(y),*)\)
• Each server sends \( S(y,z) \) to all servers \((*,h(y),h(z))\)
• Each server sends \( T(x,z) \) to all servers \((h(x),*,h(z))\)

**Final output:**

• Each server \((i,j,k)\) computes the query \( R(x,y), S(y,z), T(z,x) \) locally

**Analysis:** each tuple \( R(x,y) \) is replicated at most \( P^{\frac{1}{3}} \) times