DATA516/CSED516
Scalable Data Systems and Algorithms
Lecture 5
Parallel Query Execution
Announcements

• Project proposals were due on Friday

• Small review assignment was due today

• HW3 is posted, due on Nov. 15
Outline

• Basic notions

• Distributed query processing algorithms

• Skew (will continue next lecture)
Distributed/Parallel Query Processing

Parallel DBs since the 80s

Usually limited to small number of servers
Why?

New trend: cloud databases.
E.g. Snowflake
Distributed/Parallel Query Processing

Parallel DBs since the 80s

Usually limited to small number of servers
Why? Transactions!

New trend: cloud databases.
E.g. Snowflake
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing
Shared Memory

- SMP = symmetric multiprocessor
- Nodes share RAM and disk
- 10x … 100x processors

- Example: SQL Server runs on a single machine and can leverage many threads to speed up a query

- Easy to use and program
- Expensive to scale
Shared Disk

- All nodes access same disks
- 10x processors
- Example: Oracle
- No more memory contention
- Harder to program
- Still hard to scale
Shared Nothing

- Cluster of commodity machines
- Called "clusters" or "blade servers"
- Each machine: own memory & disk
- Up to x1000-x10000 nodes
- Example: redshift, spark, snowflake

Because all machines today have many cores and many disks, shared-nothing systems typically run many "nodes" on a single physical machine.

- Easy to maintain and scale
- Most difficult to administer and tune.
Performance Metrics

Nodes = processors = computers

• **Speedup:**
  – More nodes, same data $\Rightarrow$ higher speed

• **Scaleup:**
  – More nodes, more data $\Rightarrow$ same speed

Warning: sometimes *Scaleup* is used to mean *Speedup*
Linear v.s. Non-linear Speedup

Speedup

# nodes (\(=P\))

Ideal
Linear v.s. Non-linear Scaleup

Batch Scaleup

Ideal

# nodes (=P) AND data size

×1  ×5  ×10  ×15
Why Sub-linear?

- **Startup cost**
  - Cost of starting an operation on many nodes

- **Interference**
  - Contention for resources between nodes

- **Skew**
  - Slowest node becomes the bottleneck
“Scalability but at what cost?”

![Graph showing speedup vs. number of nodes]

- Ideal
- Best single-server algorithm

<table>
<thead>
<tr>
<th># nodes (=P)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>×1</td>
<td>×1</td>
</tr>
<tr>
<td>×5</td>
<td>×5</td>
</tr>
<tr>
<td>×10</td>
<td>×10</td>
</tr>
<tr>
<td>×15</td>
<td>×15</td>
</tr>
</tbody>
</table>
Discussion

Parallel/distributed data processing:

• Scales up* to more data:
  – More servers can hold more data

• Speedup w/ number of nodes:
  – Harder to achieve
  – But can get there with very large p

* “Scale-up” is often used informally, like here
More Discussion

New terminology:

• **Scale-up** = speedup w/ shared memory

• **Scale-out** = more data w/ more nodes

Acknowledges that speed comes from shared memory, capacity for large data comes from shared nothing
Outline

• Basic notions

• Distributed query processing algorithms

• Skew (will continue next lecture)
Distributed Query Processing Algorithms
Horizontal Data Partitioning

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table
Horizontal Data Partitioning

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table

R
Horizontal Data Partitioning

Table

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R

R₁

R₂

R₃

...
Horizontal Data Partitioning

• Block Partition, a.k.a. Round Robin:
  – Partition tuples arbitrarily s.t. size(R_1) ≈ ... ≈ size(R_P)

• Hash partitioned on attribute A:
  – Tuple t goes to chunk i, where i = h(t.A) mod P + 1

• Range partitioned on attribute A:
  – Partition the range of A into -∞ = v_0 < v_1 < ... < v_P = ∞
  – Tuple t goes to chunk i, if v_{i-1} < t.A < v_i
Notations

$p = \text{number of servers (nodes) that hold the chunks}$

When a relation $R$ is distributed to $p$ servers, we draw the picture like this:

$$R = R_1 \cup R_2 \cup \cdots \cup R_p$$

Here $R_1$ is the fragment of $R$ stored on server 1, etc.
Uniform Load and Skew

• \(|R| = N\) tuples, then \(|R_1| + |R_2| + \ldots + |R_p| = N\)

• We say the load is uniform when:
  \(|R_1| \approx |R_2| \approx \ldots \approx |R_p| \approx N/p\)

• Skew means that some load is much larger:
  \(\max_i |R_i| >> N/p\)

We design algorithms for uniform load, discuss skew later
Parallel Algorithm

• Selection $\sigma$

• Join $\Join$

• Group by $\gamma$
Parallel Selection

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

**Query:** $\sigma_{A=v}(R)$, or $\sigma_{v_1<A<v_2}(R)$

- **Block partitioned:**

- **Hash partitioned:**

- **Range partitioned:**
Parallel Selection

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

**Query:** \( \sigma_{A=v}(R) \), or \( \sigma_{v_1<A<v_2}(R) \)

- **Block partitioned:**
  - All servers need to scan
- **Hash partitioned:**
- **Range partitioned:**

**Range partitioned:**
Parallel Selection

Data: \( R(K, A, B, C) \)
Query: \( \sigma_{A=v}(R) \), or \( \sigma_{v_1<A<v_2}(R) \)

- Block partitioned:
  - All servers need to scan
- Hash partitioned:
  - Point query: only one server needs to scan
  - Range query: all servers need to scan
- Range partitioned:
Parallel Selection

Data: \( R(K, A, B, C) \)
Query: \( \sigma_{A=v}(R) \), or \( \sigma_{v_1<A<v_2}(R) \)

- Block partitioned:
  - All servers need to scan
- Hash partitioned:
  - Point query: only one server needs to scan
  - Range query: all servers need to scan
- Range partitioned:
  - Only some servers need to scan
Parallel GroupBy

Data: \( R(K, A, B, C) \)
Query: \( \gamma_{A, \text{sum}(C)}(R) \)
Discuss in class how to compute in each case:

- \( R \) is hash-partitioned on \( A \)
- \( R \) is block-partitioned or hash-partitioned on \( K \)
Parallel GroupBy

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

Discuss in class how to compute in each case:

- \( R \) is hash-partitioned on \( A \)
  - Each server \( i \) computes locally \( \gamma_{A, \text{sum}(C)}(R_i) \)
- \( R \) is block-partitioned or hash-partitioned on \( K \)
Parallel GroupBy

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

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

Discuss in class how to compute in each case:

• \( R \) is hash-partitioned on \( A \)
  – Each server \( i \) computes locally \( \gamma_{A,\text{sum}(C)}(R_i) \)

• \( R \) is block-partitioned or hash-partitioned on \( K \)
  – Need to reshuffle data on \( A \) first (next slide)
  – Then compute locally \( \gamma_{A,\text{sum}(C)}(R_i) \)
Basic Parallel GroupBy

Data: \[ R(K, A, B, C) \]
Query: \[ \gamma_{A, \text{sum}(C)}(R) \]

- \( R \) is block-partitioned or hash-partitioned on \( K \)
Basic Parallel GroupBy

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 \( A \)

\[ R_1 \quad R_2 \quad \ldots \quad R_p \]
Basic Parallel GroupBy

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

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

- \( R \) is block-partitioned or hash-partitioned on \( K \)

Reshuffle \( R \) on attribute \( A \)
Basic Parallel GroupBy

**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 \( A \)

\[
\begin{align*}
R_1' & \quad R_2' & \quad \ldots & \quad R_P' \\
R_1 & \quad R_2 & \quad \ldots & \quad R_P
\end{align*}
\]
Basic Parallel GroupBy

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

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

- \( R \) is block-partitioned or hash-partitioned on \( K \)

\[ R_1' \quad R_2' \quad \ldots \quad R_P' \]

Reshuffle \( R \) on attribute \( A \)
Basic Parallel GroupBy

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 \( A \)

This is done in \textit{one} communication step
Basic Parallel GroupBy

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 \( A \)

This is done in one communication step

Describe the push v.s. pull method
Reshuffling

• Nodes send data over the network

• Many-many communications possible

• Throughput:
  – Better than disk
  – Worse than main memory
Basic Parallel GroupBy

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

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

- \( R \) is block-partitioned or hash-partitioned on \( K \)

This is done in one communication step

Can you think of an optimization?
### GroupBy/Union Commutativity

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>LA</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>Seattle</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>NY</td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>NY</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>LA</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>Austin</td>
<td></td>
<td>55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>LA</td>
<td></td>
<td>77</td>
</tr>
<tr>
<td>NY</td>
<td></td>
<td>88</td>
</tr>
<tr>
<td>LA</td>
<td></td>
<td>99</td>
</tr>
</tbody>
</table>

```sql
SELECT city, sum(qant) FROM R GROUP BY city
```
### GroupBy/Union Commutativity

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Austin</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

Q: What is sum for Seattle?  
A: 96

SELECT city, sum(quant)  
FROM R  
GROUP BY city
GroupBy/Union Commutativity

---

**Q: What is sum for Seattle?**

**A: 106**

---

**SELECT city, sum(quant)**
FROM R
GROUP BY city
### GroupBy/Union Commutativity

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

- Sum here = 40
- Q: What is sum for Seattle? A: 106

\[
\text{SELECT city, sum(quant)} \\
\text{FROM R} \\
\text{GROUP BY city}
\]

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

- Sum here = 66
**GroupBy/Union Commutativity**

### Table 1

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

Q: What is sum for Seattle?
A: 106

### Table 2

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Austin</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>

SELECT city, sum(qant)
FROM R
GROUP BY city

### Table 3

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>qant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

\[ \gamma_{city,sum(q)}(R_1 \cup R_2 \cup R_3) = \]

Sum here = 40
Sum here = 66
### GroupBy/Union Commutativity

**SQL Query:**

```sql
SELECT city, sum(quant) 
FROM R 
GROUP BY city 
```

**Table:**

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>quant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Seattle</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>quant</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Austin</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>city</th>
<th>...</th>
<th>quant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>NY</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>

**Examples:**

- **Q:** What is sum for Seattle?  
  **A:** 106

**Commutativity:**

\[
\gamma_{\text{city}, \text{sum}(q)}(R_1 \cup R_2 \cup R_3) = \gamma_{\text{city}, \text{sum}(q)}(\gamma_{\text{city}, \text{sum}(q)}(R_1) \cup \gamma_{\text{city}, \text{sum}(q)}(R_2) \cup \gamma_{\text{city}, \text{sum}(q)}(R_3))
\]
Basic Parallel GroupBy

Data: $\mathbf{R}(K, A, B, C)$
Query: $\gamma_{A,\text{sum}(C)}(\mathbf{R})$
Basic Parallel GroupBy

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

Step 0: [Optimization] each server \( i \) computes local group-by:
\[
T_i = \gamma_{A, \text{sum}(C)}(R_i)
\]
Basic Parallel GroupBy

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

**Step 0:** [Optimization] each server \( i \) computes local group-by:
\[
T_i = \gamma_{A, \text{sum}(C)}(R_i)
\]

**Step 1:** partitions tuples in \( T_i \) using hash function \( h(A) \):
\[
T_{i,1}, T_{i,2}, \ldots, T_{i,p}
\]
then send fragment \( T_{i,j} \) to server \( j \)
Basic Parallel GroupBy

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

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

**Step 0:** [Optimization] each server \( i \) computes local group-by:

\[
T_i = \gamma_{A,\text{sum}(C)}(R_i)
\]

**Step 1:** partitions tuples in \( T_i \) using hash function \( h(A) \):

\[
T_{i,1}, T_{i,2}, \ldots, T_{i,p}
\]

then send fragment \( T_{i,j} \) to server \( j \)

**Step 2:** receive fragments, union them, then group-by

\[
R_j' = T_{1,j} \cup \ldots \cup T_{p,j}
\]

Answer \( j \) = \( \gamma_{A,\text{sum}(C)}(R_j') \)
Pushing Aggregates Past Union

Which other rules can we push past union?

• Sum?
• Count?
• Avg?
• Max?
• Median?
Pushing Aggregates Past Union

Which other rules can we push past union?

- Sum?
- Count?
- Avg?
- Max?
- Median?

<table>
<thead>
<tr>
<th>Distributive</th>
<th>Algebraic</th>
<th>Holistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum(a₁+a₂+...+a₉) = sum(sum(a₁+a₂+a₃)+ sum(a₄+a₅+a₆)+ sum(a₇+a₈+a₉))</td>
<td>avg(B) = sum(B)/count(B)</td>
<td>median(B)</td>
</tr>
</tbody>
</table>
Example Query with Group By

```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
```
Example Query with Group By

```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
\[
\text{SELECT } a, \text{sum}(b) \text{ as } sb \quad \text{FROM } R \quad \text{WHERE } c > 0 \quad \text{GROUP BY } a
\]
Speedup and Scaleup

Consider the query $\gamma_{A,\text{sum}(C)}(R)$
Assume the local runtime for group-by is linear $O(|R|)$

If we double number of nodes $P$, what is the runtime?

If we double both $P$ and size of $R$, what is the runtime?
Speedup and Scaleup

Consider the query $\gamma_{A,\sum(C)}(R)$
Assume the local runtime for group-by is linear $O(|R|)$

If we double number of nodes $P$, what is the runtime?
- Half (chunk sizes become $\frac{1}{2}$)

If we double both $P$ and size of $R$, what is the runtime?
- Same (chunk sizes remain the same)
Speedup and Scaleup

Consider the query $\gamma_{A,\text{sum}(C)}(R)$
Assume the local runtime for group-by is linear $O(|R|)$

If we double number of nodes $P$, what is the runtime?
- Half (chunk sizes become $\frac{1}{2}$)

If we double both $P$ and size of $R$, what is the runtime?
- Same (chunk sizes remain the same)

But only if the data is without skew!
Parallel/Distributed Join

Three “algorithms”:

- Hash-partitioned
- Broadcast
- Combined: “skew-join” or other names
Distributed Hash-Join
Hash Join: \( R \bowtie_{A=B} S \)

Data: \( R(A, C), S(B, D) \)

Query: \( R \bowtie_{A=B} S \)

Initially, \( R \) and \( S \) are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)
Some servers hold \( R \)-chunks, some hold \( S \)-chunks, some hold both
Hash Join: $R \bowtie_{A=B} S$

Data: $R(A, C), S(B, D)$

Query: $R \bowtie_{A=B} S$

Reshuffle $R$ on $R.A$
and $S$ on $S.B$

Initially, $R$ and $S$ are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)
Some servers hold $R$-chunks, some hold $S$-chunks, some hold both
Hash Join: \( R \bowtie_{A=B} S \)

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

Initially, \( R \) and \( S \) are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)
Some servers hold \( R \)-chunks, some hold \( S \)-chunks, some hold both

Reshuffle \( R \) on \( R.A \) and \( S \) on \( S.B \)
Hash Join: \( R \bowtie_{A=B} S \)

**Data:** \( R(A, C), S(B, D) \)

**Query:** \( R \bowtie_{A=B} S \)

- Each server computes the join locally.
- Reshuffle \( R \) on \( R.A \) and \( S \) on \( S.B \).

Initially, \( R \) and \( S \) are block partitioned.

Notice: they may be stored in DFS (recall MapReduce).

Some servers hold \( R \)-chunks, some hold \( S \)-chunks, some hold both.
Hash Join: $R \bowtie_{A=B} S$

- **Step 1**
  - Every server holding any chunk of $R$ partitions its chunk using a hash function $h(t.A)$
  - Every server holding any chunk of $S$ partitions its chunk using a hash function $h(t.B)$

- **Step 2:**
  - Each server computes the join of its local fragment of $R$ with its local fragment of $S$
Broadcast Join
A.k.a. “Small Join”
Broadcast Join

• When joining R and S
• If $|R| >> |S|$
  – Leave R where it is
  – Replicate entire S relation across R-nodes

• Called a **small join** or a **broadcast join**
Query: \( R \bowtie S \)

Broadcast Join

\[ R_1 \quad R_2 \quad R_P \quad S \]

\[ \ldots \]
Query:  \( R \bowtie S \)

Broadcast Join

Keep R in place

Broadcast S
Query: $R \bowtie S$

Broadcast Join

Same place…

Keep R in place

Broadcast S
Query: \( R \Join S \)

Broadcast Join

- Keep \( R \) in place
- Same place...
- Broadcast \( S \)

\( R_1, S \) \( \rightarrow \) \( R_2, S \) \( \rightarrow \) \( R_P, S \) \( \rightarrow \) \( S \)
Example Query Execution

Find all orders from today, along with the items ordered

SELECT *
FROM Order o, Line i
WHERE o.item = i.item
AND o.date = today()
Query Execution

Node 1

hash

h(o.item)

select
date=today()

scan

Order o

Node 2

hash

h(o.item)

select
date=today()

scan

Order o

Node 3

hash

h(o.item)

select
date=today()

scan

Order o

Join

o.item = i.item

Select
date = today()

Scan

Order o

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

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

Query Execution

Node 1

Node 2

Node 3

Join

Join

Join

o.item = i.item

o.item = i.item

o.item = i.item

contains all orders and all lines where hash(item) = 3

contains all orders and all lines where hash(item) = 2

contains all orders and all lines where hash(item) = 1

Node 1

Node 2

Node 3
Example 2

SELECT *
FROM R, S, T
WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100
... WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100
... WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100
\[\sigma_{R.a-T.f > 100}\]

... WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100
\[ \sigma_{R.a - T.f > 100} \]

\[ R \bowtie S \]

\[ \sigma_{R.a - T.f > 100} \]

\[ R \bowtie S \]

\[ \sigma_{R.a - T.f > 100} \]

\[ R \bowtie S \]

Shuffling intermediate result from \( R \bowtie S \)

\[ h(S.d) \]

\[ R \bowtie S \]

Shuffling \( R, S, \) and \( T \)

\[ h(R.b) \]

\[ h(S.c) \]

\[ h(T.e) \]

\[ h(R.b) \]

\[ h(S.c) \]

\[ h(T.e) \]

\[ h(R.b) \]

\[ h(S.c) \]

\[ h(T.e) \]

\[ h(R.b) \]

\[ h(S.c) \]

\[ h(T.e) \]

... WHERE \( R.b = S.c \) AND \( S.d = T.e \) AND \( (R.a - T.f) > 100 \)

Machine 1

1/3 of \( R, S, T \)

Machine 2

1/3 of \( R, S, T \)

Machine 3

1/3 of \( R, S, T \)
\[\sigma_{R.a - T.f > 100}\]

\[R \bowtie S \bowtie T\]

\(\sigma_{R.a - T.f > 100}\)  \[R \bowtie S \bowtie T\]  \[R \bowtie S \bowtie T\]

broadcast  broadcast  broadcast

scan S  scan S  scan S

scan T  scan T  scan T

Machine 1  Machine 2  Machine 3

1/3 of R, S, T  1/3 of R, S, T  1/3 of R, S, T
Discussion

• Hash-join:
  – Both relations are partitioned \((\text{good})\)
  – May have skew \((\text{bad})\)
Discussion

• Hash-join:
  – Both relations are partitioned (good)
  – May have skew (bad)

• Broadcast join
  – One relation must be broadcast (bad)
  – No worry about skew (good)
Discussion

• Hash-join:
  – Both relations are partitioned (good)
  – May have skew (bad)

• Broadcast join
  – One relation must be broadcast (bad)
  – No worry about skew (good)

• Skew join (has other names):
  – Combine both: in class
Outline

• Basic notions

• Distributed query processing algorithms

• Skew (will continue next lecture)
Skew
Skew

• Skew means that one server runs much longer than the other servers

• Reasons:
  – Computation skew
  – Data skew
Computation Skew

- All workers receive the same amount of input data, but some need to run much longer than others
- E.g. perform some image processing whose runtimes depends on the image
- Solution: use virtual servers
Virtual Servers

Main idea:

• If we send the data uniformly to the P servers, and one of them is stuck with the complicated image, then we have skew

• Solution: pretend we have many “virtual” servers. (Next slide.)
Virtual Servers

Large number $P_v$ of “virtual servers”

• Design algorithm for $P_v$ virtual servers

• Scale down to $P \ll P_v$ physical servers, by simulating them round-robin

E.g. MapReduce: $P$=workers, $P_v$=map tasks
Data Skew

• We fail to distribute the data uniformly to the servers
• Question: why can this happen?
Data Skew

• We fail to distribute the data uniformly to the servers

• Question: why can this happen?

• Answer:
  – Range partition may have many more tuples in one bucket than another
  – Hash partition may suffer from heavy hitters
Next Lecture

• Analyze skew: notice hw3 question

• New topic: scalable graph processing