CSE 344

AUGUST 15TH

MORE PARALLEL QUERY OPTIMIZATION
AND AN ADVANCED TOPIC
ADMINISTRIVIA

• HW8 due tonight

• Course evaluations!
  • (may help your grade if participation is high)

• Section tomorrow: exam review
  • my notes on what is important
EXAM

• Friday, in class

• Similar to midterm
  • designing for 1 hour
  • can go over if necessary

• Note sheet allowed
  • one page, both sides
  • cost formulas will be provided
EXAM

• Four questions
  1. parallel databases (including today)
  2. database design: E/R & normalization
  3. transactions
  4. multiple choice / short answer
     • references to 1st half material sprinkled throughout

• Preparation
  • practice exams on web
  • lecture videos will be made available tonight
DISTRIBUTED QUERY PROCESSING

Parallel DBs storing data that is partitioned across machines

- OLTP is still easy
- OLAP more difficult

We look at this before in terms of cost of disk I/O

- in general, time multiplied by 1 / #machines (speed up)

Today: network cost

- partially to review for final
- partially because network cost is increasingly relevant in modern systems (disks are too slow)
DISTRIBUTED QUERY PROCESSING

Data is horizontally partitioned on many servers

- ideally, stored in memory
- disk cost ≫ network cost ≫ memory cost ≫ CPU cost
  - if the query hits disk, that likely dominates all other costs
- storing in memory means a huge reduction in cost
  - memory is cheap enough that companies can do this
HORIZONTAL DATA PARTITIONING

Data:

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Servers:

1

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... 

P

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which tuples go to what server?
HORIZONTAL DATA PARTITIONING

Data:

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Servers:

1

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

2

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

... P

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

assume block or hash partitioning
DISTRIBUTED QUERY PROCESSING

Data is horizontally partitioned on many servers
- stored in memory
- main cost is now network cost
  - network cost $\gg$ memory cost $\gg$ CPU cost

Operators may require data reshuffling
- move data to the machines that needs it
- this is the only new element in parallel query processing
- this is also the only part with network cost!
  - everything else is too small to matter

Still measure cost in blocks
- like disk, network cost is proportional to size of data sent
- (blocks have size in bytes, tuples do not)
PARALLEL EXECUTION OF RA OPERATORS: SELECTION

Data: $R(K, A, B, C)$
Query: $\sigma_{A=c}(R)$

No change necessary
- Send query to every machine
- Each sends back its tuples that satisfy selection
- Result is the union of these

Cost: $B(\sigma_{A=c}(R)) / P$

$B(\sigma_{A=c}(R))$ is total size
BUT data is sent in parallel
PARALLEL EXECUTION OF RA OPERATORS: SELECTION

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

**Query:** \( \sigma_{A=c}(R) \)

No change necessary

- Send query to every machine
- Each sends back its tuples that satisfy selection
- Result is the union of these

\[ \text{Cost: } \frac{B(\text{output})}{P} \]

Simplification write “output” for \( \sigma_{A=c}(R) \)
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 \( K \)

Cost: \( \frac{B(R)}{P} + \frac{B(\text{output})}{P} \)

network cost of reshuffle and sending output

Reshuffle \( R \) on attribute \( A \)

Run grouping on reshuffled partitions
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 $A$

**Cost:** $B(\text{output}) / P$
PARALLEL EXECUTION OF RA OPERATORS: PARTITIONED HASH-JOIN

Data: $R(K_1, A, B), S(K_2, B, C)$

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

- Initially, both $R$ and $S$ are partitioned on $K_1$ and $K_2$

Cost: $\frac{B(R)}{P} + \frac{B(S)}{P} + \frac{B(output)}{P}$

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

- Each server computes the join locally
DISTRIBUTED QUERY PROCESSING

So far cost is

- $\frac{B(\text{output})}{P}$
  - this is never going away
- plus $\frac{B(R)}{P}$ for any $R$ that needs a reshuffle
  - in a join, only one of the two parts may need a reshuffle

Not every case looks like that...
**BROADCAST JOIN**

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

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

*Cost:* \( B(S) + B(output) / P \)

- \( R \) does not need to move!
- \( B(S) / P \) becomes \( B(S) \)
  - BUT we drop \( B(R) / P \)
  - \( R \) does not need to move!
**BROADCAST JOIN**

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

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

Cost: \( B(S) + B(output) / P \)

When would that be better?

Diagram:

- \( R_1 \)
- \( R_2 \)
- \( \ldots \)
- \( R_P \)
- \( S \)

\( R_1, S \)
\( R_2, S \)
\( \ldots \)
\( R_P, S \)
Would there ever be a reason not to push selections down?

• common heuristic even in non-distributed query optimization

I can’t see one

• can only reduce the amount of data we need to shuffle
• why didn’t we always do this with disks?
  • can lose our ability to do an indexed selection
  • we have an index on R not $\sigma_{A=c}(R)$
DISTRIBUTED QUERY PROCESSING

What is still missing compared to non-distributed case?

Indexes

• not much of a help here!
• (think about it on your own sometime)

(Things don’t always get more complex in a better system!)
DISTRIBUTED QUERY OPTIMIZATION

Not any harder (maybe easier) than non-distributed case

Still not trivial

- different physical plans: broadcast vs shuffling joins
- different logical plans: join orders
  - e.g., \((R \bowtie S) \bowtie T\) vs \(R \bowtie (S \bowtie T)\)
    - both shuffle R, S, and T
    - but first has extra shuffle of \(R \bowtie S\), the other of \(S \bowtie T\)
  - this is a big part of non-distributed query opt also
EXAMPLE

Compare two logical plans

- \((R \bowtie S) \bowtie T\)
- \(R \bowtie (S \bowtie T)\)

With different physical plans

- first: broadcast \(R\) in first join, reshuffle in second
- second: reshuffles all around

Suppose they are initially partitioned as follows

- \(R\) is partitioned on \(A\)
- \(S\) and \(T\) are block partitioned

\[
\begin{align*}
R & (A, B) \\
S & (B, C) \\
T & (A, C)
\end{align*}
\]
EXAMPLE

Ignore the output cost

- it is the same for both plans
- just look at reshuffling costs

Cost of \((R \bowtie S) \bowtie T\)

- first join: \(R \bowtie S\), broadcasting \(R\)
  - cost is \(B(R)\)
  - no factor of \(1/P\) since each machine gets all of \(R\)
- second join: \((R \bowtie S) \bowtie T\), reshuffling both
  - cost is \(B(R \bowtie S)/P + B(T)/P\)
  - total cost is \(B(R) + B(T)/P + B(R \bowtie S)/P\)
EXAMPLE

Cost of $R \bowtie (S \bowtie T)$

- first join: $S \bowtie T$, reshuffling both
  - cost is $B(S)/P + B(T)/P$
- second join: $R \bowtie (S \bowtie T)$, reshuffling only $S \bowtie T$
  - why? (recall that $R$ is initially partitioned on $A$)
    - equijoin on $A$ & $B$...
    - need tuples with the same value of $A$ & $B$ on same machine
    - $R$ is already partitioned by $A$ so...
      - tuples of $R$ with same value of $A$ already on same machine
        - including the ones that also have same value of $B$
    - cost is $B(S \bowtie T)/P$
- total cost is $B(S)/P + B(T)/P + B(S \bowtie T)/P$
EXAMPLE

Costs

- \((R \bowtie S) \bowtie T\) \quad B(R) + B(T)/P + B(R \bowtie S)/P
- \(R \bowtie (S \bowtie T)\) \quad B(S)/P + B(T)/P + B(S \bowtie T)/P

Which is faster?

Need to estimate sizes of \(R \bowtie S\) and \(S \bowtie T\)

- How do we do that?
  - selectivity (same as before)
  - let \(E\) be the selectivity of \(=\) on \(R.B\)
  - let \(F\) be the selectivity of \(=\) on \(S.C\)
- \(R \bowtie S\) increases size of \(S\) by \(T(R)/E\), so \(T(R)B(S)/E\)
- \(S \bowtie T\) increases size of \(T\) by \(T(S)/F\), so \(T(S)B(T)/F\)
**EXAMPLE**

Costs

- $(R \bowtie S) \bowtie T$: $B(R) + B(T)/P + T(R)B(S)/PE$
- $R \bowtie (S \bowtie T)$: $B(S)/P + B(T)/P + T(S)B(T)/PF$

Which is faster?

- When is $B(R) + T(R)B(S)/PE > B(S)/P + T(S)B(T)/PF$?
- Plug in the numbers $B(...)$, $T(..)$, $E$ ,and $F$ to find out
- Some observation though...
  - left side uses $B(R)$ and $T(R)$ while right side has neither
  - second plan will be much faster when $R$ is large
  - first plan broadcasts $R$, so it wants $R$ to be small
  - second plan doesn’t even need to shuffle $R$, so no cost
EXAMPLE

Costs

- \((R \bowtie S) \bowtie T)\quad \text{B(R)} + \frac{\text{B(T)}}{P} + \frac{T(R)\text{B(S)}}{\text{PE}}
- \(R \bowtie (S \bowtie T)\quad \frac{\text{B(S)}}{P} + \frac{\text{B(T)}}{P} + \frac{T(S)\text{B(T)}}{\text{PF}}

Which is faster?

- When is \(\frac{\text{B(R)} + \frac{T(R)\text{B(S)}}{\text{PE}}}{\text{B(S)}} > \frac{\text{B(S)}}{P} + \frac{T(S)\text{B(T)}}{\text{PF}}\) ?
- Plug in the numbers \(\text{B(...)}, \text{T(..)}, \text{E },\text{and F to find out}
- Some observation though...
  - second plan will be much faster when \(R\) is large
  - first plan broadcasts \(R\), so it wants \(R\) to be small
  - second plan doesn’t even need to shuffle \(R\), so no cost
  - right side uses \(\text{B(T)}\) while left side does not (nor \(\text{T(T)}\))
  - second plan shuffles \(S \bowtie T\), so it wants \(T\) to be small
  - first plan will be much faster when \(T\) is large
NETWORK COST FORMULAS

(All ignore output cost)

\( \sigma \)  
free

\( \pi \)  
free

\( \gamma \ldots (R) \)  
\( B(R) / P \)

\( R \bowtie S \)  

- shuffle \( R \) and \( S \)  
\( B(R) / P + B(S) / P \)
- shuffle \( R \) only  
\( B(R) / P \)
- shuffle \( S \) only  
\( B(S) / P \)
- broadcast \( R \)  
\( B(R) \)
- broadcast \( S \)  
\( B(S) \)

\( Q_1 \cup Q_2 \)  
\( \text{cost of } Q_1 + \text{cost of } Q_2 \)

\( R - S \)  
exercise!
LAST TOPIC
(ADVANCED)
Imagine a table with rows for individuals. Is there a way to analyze the group while preserving the privacy of individuals?

- e.g., can I determine whether one subset of the individuals differs from another subset without leaking details of any individuals?
Is there a way to analyze the group while preserving the privacy of individuals?

How do we even define this?

- say the analysis is privacy-preserving if changing the tuple for any individual does not change results
  - if the analysis was capturing information about them, then the results would change
- unfortunately, we can’t do this exactly...
DIFFERENTIAL PRIVACY

We can do something similar (in many cases)

• analysis will involve random choices
• want: probability result changes is $< \epsilon$ when any individual record is changed
  • (probability over random choices in analysis)
• this is differential privacy (modulo details)
• randomization is essential here
DIFFERENTIAL PRIVACY: PRACTICAL EXAMPLE

Find fraction of people with bad property P
  • people don’t want it known if they have P

Collect data with this mechanism
  • for each person, flip a coin
    • if heads, answer truthfully
    • if false, answer Yes/No randomly (50/50%)
  • those answering Yes have “plausible deniability”
  • if P percent say yes, true answer is $2P - \frac{1}{4}$
    • adjusts for random Yes’s without property P
DIFFERENTIAL PRIVACY

Invented by Dwork and McSherry (2005)

• fixed problems in earlier work on “anonymization”
  • e.g., people were able to identify Netflix users from the data Netflix made available to researchers
• uses same idea as previous: add randomness to data
• won the Gödel prize (and others)
• works for many but not all types of queries

Could be applied to a wide range of problems

• e.g., an app to analyze usage trends without seeing every detail of user activity