CSE 344

JULY 23\textsuperscript{RD}

PARALLEL DATABASES
ADMINISTRIVIA

• HW5 due Wednesday

• Sign up for Amazon credits
  • need for HW6. can take a while

• Midterm on Friday
  • Practice exam on web site
  • Videos from last 2 weeks all on web site
  • No need to memorize cost formulas but do need to understand them
WHY COMPUTE IN PARALLEL?

Multi-cores:

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

Big data: too large to fit in main memory

- Disk has more space but is slow
- Distributed query processing on 100x-1000x servers
- Widely available now using cloud services
PERFORMANCE METRICS FOR PARALLEL DBS

Nodes = processors or computers

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

**Scaleup:**
- More nodes, more data $\rightarrow$ same speed
WHY SUB-LINEAR SPEEDUP AND SCALEUP?

Startup cost

- Cost of starting an operation on many nodes

Interference

- Contention for resources between nodes

Skew

- Slowest node becomes the bottleneck
ARCHITECTURES FOR PARALLEL DATABASES

Shared memory

Shared disk

Shared nothing
Nodes share both RAM and disk
Dozens to hundreds of processors

Example: SQL Server runs on a single machine and can leverage many threads to speed up a query
check your HW3 query plans

Easy to use and program
Expensive to scale
  • last remaining cash cows in the hardware industry
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

Hard to scale: existing deployments typically have fewer than 10 machines
Cluster of commodity machines on high-speed network
Called "clusters" or "blade servers"
Each machine has its own memory and disk: lowest contention.

Example: Google, Microsoft Cloud

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

We discuss only Shared Nothing in class

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?

We study only intra-operator parallelism: most scalable
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
Given relations $R(A,B)$ and $S(B, C)$, no indexes:

**Selection:** $\sigma_{A=123}(R)$
- Scan file $R$, select records with $A=123$

**Group-by:** $\gamma_{A,\text{sum}(B)}(R)$
- Scan file $R$, insert into a hash table using $A$ as key
- When a new key is equal to an existing one, add $B$ to the value

**Join:** $R \bowtie S$
- Scan file $S$, insert into a hash table using $B$ as key
- Scan file $R$, probe the hash table using $B$
**HORIZONTAL DATA PARTITIONING**

<table>
<thead>
<tr>
<th>Data:</th>
<th>Servers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(K)</td>
<td>1</td>
</tr>
<tr>
<td>(A)</td>
<td>2</td>
</tr>
<tr>
<td>(B)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>1</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>2</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(P)</td>
</tr>
</tbody>
</table>
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

Block Partition:
- Partition tuples arbitrarily s.t. size(R₁) ≈ ... ≈ 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’s is free in this class

Range partitioned on attribute A:
- Partition the range of A into -∞ = v₀ < v₁ < ... < vₚ = ∞
- Tuple t goes to chunk i, if vᵢ₋₁ < t.A ≤ vᵢ
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 $A$

**Range partition**
- May be skewed

Assuming good hash function and mode is not too large

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: $y_{A,\text{sum}(C)}(R)$
  • Runtime: only consider Disk 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!
SKEWED DATA

• \( R(K,A,B,C) \)
• 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 many times
• Then the server holding that value will be skewed
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 \( K_2 \)

Reshuffle \( R \) on \( R.B \) and \( S \) on \( S.B \)

Each server computes the join locally
Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) \bowtie S(K2, B, C)

PARALLEL JOIN ILLUSTRATION

Partition

Shuffle on B

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

Why would you want to do this?
EXAMPLE PARALLEL QUERY PLAN

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()
```
Order(oid, item, date), Line(item, …)

PARALLEL QUERY PLAN

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
PARALLEL QUERY PLAN

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

**EXAMPLE PARALLEL QUERY PLAN**

- **Node 1**: Contains all orders and all lines where hash(item) = 1
- **Node 2**: Contains all orders and all lines where hash(item) = 2
- **Node 3**: Contains all orders and all lines where hash(item) = 3

Join conditions:
- in Node 1: o.item = i.item
- in Node 2: o.item = i.item
- in Node 3: o.item = i.item