Lecture 12 – Parallel DBMSs
Announcements

• HW3 due on Friday!!!

• HW4 posted please apply for AWS credits asap

• Project, project, project, ...

• No class on Monday
Where We Are

• Relational data model: SQL, RA, datalog, FDs, ...

• Systems: disk I/Os, buffer, physical RA, iterator model, ...

• Today: scaling up to parallel computation
Two Ways to Scale a DBMS

Scale up

Scale out

A more powerful server

More servers
Two Ways to Scale a DBMS

• Obviously this can be used to:
  – Execute multiple queries in parallel
  – Speed up a single query

• For now: how to speed up a single query

• We will worry about how to scale to multiple queries later
Parallel v.s. Distributed Databases

• **Distributed database system:**
  – Data is managed by **several sites**, each site capable of running independently

• **Parallel database system:**
  – Data is managed by a **single site**, but processed distributively, using parallel implementation
Parallel DBMSs

• **Goal**
  – Improve performance by executing multiple operations in parallel

• **Key benefit**
  – Cheaper to scale than relying on a single increasingly more powerful processor

• **Key challenge**
  – Ensure overhead and contention do not kill performance
Performance Metrics for Parallel DBMSs

Speedup

- More processors $\Rightarrow$ higher speed
- Individual queries should run faster
- Should do more transactions per second (TPS)
- Fixed problem size overall, vary # of processors ("strong scaling")
Linear v.s. Non-linear Speedup

![Graph showing linear and non-linear speedup]

ideal

for real...

Speedup

# processors (=P)

CSE 544 - Winter 2018
Performance Metrics for Parallel DBMSs

Scaleup

- More processors ➔ can process more data
- Fixed problem size per processor, vary # of processors ("weak scaling")
- Batch scaleup
  - Same query on larger input data should take the same time
- Transaction scaleup
  - N-times as many TPS on N-times larger database
  - But each transaction typically remains small
Linear v.s. Non-linear Scaleup

Batch Scaleup

# processors (=P) AND data size

ideal

× 1

× 5 for real…

× 10

× 15

CSE 544 - Winter 2018
Buzzwords, buzzwords

• Be careful. Commonly used terms today:
  – “scale up” = use an increasingly more powerful server
  – “scale out” = use a larger number of servers
Challenges to Linear Speedup and Scaleup

• **Startup cost**
  – Cost of starting an operation on many processors

• **Interference**
  – Contention for resources between processors

• **Skew**
  – Slowest processor becomes the bottleneck
Parallel DBMS Architectures
Architecture for Parallel DBMS: Shared Memory

Interconnection Network

Global Shared Memory

Aka SMP = symmetric multi processor
Architecture for Parallel DBMS: Shared Disk
Architecture for Parallel DBMS: Shared Nothing

Interconnection Network

P
M
D

P
M
D

P
M
D
A Professional Picture…

From: Greenplum Database Whitepaper

SAN = “Storage Area Network”
Shared Memory

• Nodes share both RAM and disk
• Dozens to hundreds of processors

Example: SQL Server runs on a single machine
• leverage many threads to get a query to run faster

Characteristics:
• Easy to use and program
• But very expensive to scale
Shared Disk

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

Oracle dominates this class of systems

Characteristics:
• Also hard to scale past a certain point: existing deployments typically have fewer than 10 machines
Shared Nothing

- Cluster of machines on high-speed network
- Called "clusters" or "blade servers"
- Each machine has its own memory and disk: lowest contention.

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

Characteristics:
- Today, this is the most scalable architecture.
- Most difficult to administer and tune.

We discuss only Shared Nothing in class
So…

• You have a parallel machine. Now what?

• How do you speed up your DBMS given a shared-nothing architecture?
Approaches to Parallel Query Evaluation

• Inter-query parallelism
  – Each query runs on one processor
  – Only for running multiple queries (OLTP)
Approaches to Parallel Query Evaluation

- **Inter-query parallelism**
  - Each query runs on one processor
  - Only for running multiple queries (OLTP)

- **Inter-operator parallelism**
  - A query runs on multiple processors
  - An operator runs on one processor
  - For both OLTP and Decision Support
Approaches to Parallel Query Evaluation

- **Inter-query parallelism**
  - Each query runs on one processor
  - Only for running multiple queries (OLTP)

- **Inter-operator parallelism**
  - A query runs on multiple processors
  - An operator runs on one processor
  - For both OLTP and Decision Support

- **Intra-operator parallelism**
  - An operator runs on multiple processors
  - For both OLTP and Decision Support
Approaches to Parallel Query Evaluation

- **Inter-query parallelism**
  - Each query runs on one processor
  - Only for running multiple queries (OLTP)

- **Inter-operator parallelism**
  - A query runs on multiple processors
  - An operator runs on one processor
  - For both OLTP and Decision Support

- **Intra-operator parallelism**
  - An operator runs on multiple processors
  - For both OLTP and Decision Support

We study only intra-operator parallelism: most scalable
Data Partitioning
Horizontal Data Partitioning

• Relation R split into P chunks \( R_0, \ldots, R_{P-1} \), stored at the P nodes

• Block partitioned
  – Each group of \( k \) tuples go to a different node

• Hash based partitioning on attribute \( A \):
  – Tuple \( t \) to chunk \( h(t.A) \mod P \)

• Range based partitioning on attribute \( A \):
  – Tuple \( t \) to chunk \( i \) if \( v_{i-1} < t.A < v_i \)

Need to worry about data skew
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**
- **Hash-partition**
  - On the key \( K \)
  - On the attribute \( A \)
- **Range-partition**
  - On the key \( K \)
  - On the attribute \( A \)

Assuming uniform hash function

- Uniform
  - E.g. when all records have the same value of the attribute \( A \), then all records end up in the same partition
  - Difficult to partition the range of \( A \) uniformly.

May be skewed
All You Need to Know About Skew

Hash-partition a $m$ data values (with duplicates!) to $p$ bins

**Fact 1** Expected size of any *one* fixed bin is $m/p$

**Fact 2** Say that data is *skewed* if some value has degree $> m/p$. Then *some* bin has load $> m/p$

**Fact 3** Conversely, if the database is *skew-free* then max size of *all* bins $= O(m/p)$ w.h.p.
Parallelizing Operator Implementations
Parallel Selection

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

On a conventional database: cost = $B(R)$

Q: What is the cost on a parallel database with $P$ processors?

• Block partitioned
• Hash partitioned
• Range partitioned
Parallel Selection

Q: What is the cost on a parallel database with P nodes?

A: \( \frac{B(R)}{P} \) in all cases (except range) if cost is response time

However, not all processors are equal (workwise):
- Block: all servers do the same amount of work
- Hash: one server for \( \sigma_{A=v}(R) \), all for \( \sigma_{v_1<A<v_2}(R) \)
- Range: some servers only
Parallel Group By: \( \gamma_A, \text{sum}(B)(R) \)

- If \( R \) is partitioned on \( A \), then each node computes the group-by locally.
- Otherwise, hash-partition \( R(K,A,B,C) \) on \( A \), then compute group-by locally:

\[
\begin{align*}
R_1 & \rightarrow R_2 & \ldots & \rightarrow R_P \\
R_1' & \rightarrow R_2' & \ldots & \rightarrow R_P'
\end{align*}
\]

Reshuffle \( R \) on attribute \( A \)
Parallel Group By: $\gamma_{A, \text{sum}(B)}(R)$

- Step 1: server $i$ partitions chunk $R_i$ using a hash function $h(t.A) \mod P$: $R_{i0}, R_{i1}, \ldots, R_{i,P-1}$ (there are $P$ servers total)

- Step 2: server $i$ sends partition $R_{ij}$ to server $j$

- Step 3: server $j$ computes $\gamma_{A, \text{sum}(B)}$ on $R_{0j}, R_{1j}, \ldots, R_{P-1,j}$
Parallel Group By: $\gamma_A, \text{sum}(B)(R)$

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?
Parallel Group By: \( \gamma_A, \text{sum}(B)(R) \)

- \( \text{Sum}(B) = \text{Sum}(B_0) + \text{Sum}(B_1) + \ldots + \text{Sum}(B_n) \)
- \( \text{Count}(B) = \text{Count}(B_0) + \text{Count}(B_1) + \ldots + \text{Count}(B_n) \)
- \( \text{Max}(B) = \text{Max} \left( \text{Max}(B_0), \text{Max}(B_1), \ldots, \text{Max}(B_n) \right) \)
  - \( \text{distributive} \)
- \( \text{Avg}(B) = \frac{\text{Sum}(B)}{\text{Count}(B)} \)
  - \( \text{algebraic} \)
- \( \text{Median}(B) = ??? \)
  - \( \text{holistic} \)
Parallel Join: $R \bowtie_{A=B} S$

- **Step 1**
  - For all servers in $[0,k]$, server $i$ partitions chunk $R_i$ using a hash function $h(t.A) \mod P$: $R_{i0}, R_{i1}, \ldots, R_{i,P-1}$
  - For all servers in $[k+1,P]$, server $j$ partitions chunk $S_j$ using a hash function $h(t.A) \mod P$: $S_{j0}, S_{j1}, \ldots, S_{j,P-1}$

- **Step 2:**
  - Server $i$ sends partition $R_{iu}$ to server $u$
  - Server $j$ sends partition $S_{ju}$ to server $u$

- **Steps 3:** Server $u$ computes the join of $R_{iu}$ with $S_{ju}$
Example of Parallel Query Plan

*Find all orders from today, along with the items ordered*

```
SELECT * 
FROM Orders o, Lines i 
WHERE o.item = i.item 
AND o.date = today()
```
Example Parallel Plan

Node 1

- `hash`
- `select` `date=today()`
- `scan` Order o

Node 2

- `hash`
- `select` `date=today()`
- `scan` Order o

Node 3

- `hash`
- `select` `date=today()`
- `scan` Order o

Join: `o.item = i.item`
Example Parallel Plan

Node 1

hash
h(i.item)

scan
Item i

Node 2

hash
h(i.item)

scan
Item i

Node 3

hash
h(i.item)

scan
Item i

join
o.item = i.item

date = today()

Order o

Node 1

Node 2

Node 3

CSE 544 - Winter 2018
Example Parallel Plan

- **Node 1**
  - join
  - o.item = i.item
  - contains all orders and all lines where hash(item) = 1

- **Node 2**
  - join
  - o.item = i.item
  - contains all orders and all lines where hash(item) = 2

- **Node 3**
  - join
  - o.item = i.item
  - contains all orders and all lines where hash(item) = 3

CSE 544 - Winter 2018
Optimization for Small Relations

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

• Sometimes called a “small join” or “broadcast join”
Other Interesting Parallel Join Implementation

Problem of skew during join computation

Some join partitions get more input tuples than others

• Reason 1: Base data unevenly distributed
  – Because used a range-partition function
  – Or used hashing but some values are very popular (Skew)
• Reason 2: Selection before join with different selectivities
• Reason 3: Input data got unevenly rehashed (or otherwise repartitioned before the join)

Some partitions output more tuples than others
Some Skew Handling Techniques

1. Use range- instead of hash-partitions
   - Ensure that each range gets same number of tuples
   - Example: \{1, 1, 1, 2, 3, 4, 5, 6\} → [1,2] and [3,6]

2. Create more partitions than nodes
   - And be smart about scheduling the partitions

3. Use subset-replicate (i.e., “skewedJoin”)
   - Given an extremely common value ‘v’
   - Distribute R tuples with value v randomly across k nodes (R is the build relation)
   - Replicate S tuples with value v to same k machines (S is the probe relation)
Parallel Dataflow Implementation

Use relational operators unchanged

Add a special *shuffle* operator

- Handle data routing, buffering, and flow control
- Inserted between consecutive operators in the query plan
- Two components: ShuffleProducer and ShuffleConsumer
- Producer pulls data from operator and sends to \( n \) consumers
  - Producer acts as driver for operators below it in query plan
- Consumer buffers input data from \( n \) producers and makes it available to operator through `getNext` interface
Conclusion

• Making databases parallel is another way to speed up query processing

• Many algorithms for parallelizing different relational operators

• Next time: MapReduce and Spark