Introduction to Data Management

Parallel Processing

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle
Course Context

- Core RDBMS
  - SQL and RA
  - Logical and Physical Database Design
  - Transactions

- Misc. RDBMS Topics
  - Distributed Relational Databases
  - Spark query language
  - Datalog query language

- NoSQL
Humans have a tendency to tackle problems that are too big to compute

- Breaking the enigma code (WWII)
  - Using automation (the bombe)
- Computing rocket trajectories (Space Race)
  - Using programming languages (FORTRAN)
- Now: Data driven applications
  - Protein folding
  - Internet of things
  - Financial forecasting
  - Weather prediction
  - Social media platforms
  - ...
The rates at which we generate and use information have **outpaced the capabilities of a single computer**

**Problems:**
- Need more speed
- Need more scale
Parallel Computation

- Solution: Add more computing nodes
  - Multiple nodes → Parallel data management
- Most all computers have **multiple cores**
- Distributed architecture is easily available on **cloud services**
**Speed up:**
same data, more nodes $\Rightarrow$ higher speed

![Diagram showing query speed vs. number of computing nodes](image)

- **Ideal-linear speedup**

---

**November 15, 2019**

Parallel Processing
Scale up:

more data, more nodes \( \square \) same speed
Sublinear Expected Performance

- Parallel computing is not a magic bullet
- Common reasons for sublinear performance:
  - **Overhead cost**
    - Starting and coordinating operations on many nodes
  - **Interference/Contention**
    - Shared resources are not perfectly split
  - **Skew**
    - Process is only as fast as the slowest node
Implementations for Database Parallelism

- **Architecture Parallelism**
  - Shared Memory
  - Shared Disk
  - Shared Nothing*

- **Query Parallelism**
  - Inter-Query Parallelism
  - Intra-Query Parallelism
    - Inter-Operator Parallelism
    - Intra-Operator Parallelism*
Implementations for Database Parallelism

- Architecture Parallelism
  - Shared Memory
  - Shared Disk
  - Shared Nothing*

- Query Parallelism
  - Inter-Query Parallelism
  - Intra-Query Parallelism
    - Inter-Operator Parallelism
    - Intra-Operator Parallelism*
Shared-Memory Architecture

- Shared main memory and disks
- Your laptop or desktop uses this architecture
- Expensive to scale
- Easiest to implement on

Shared main memory and disks

Interconnection Network (Motherboard)

Global Memory

D  D  D

D  D  D

Your laptop or desktop uses this architecture

Expensive to scale

Easiest to implement on

Microsoft SQL Server

PostgreSQL

SQLite

MySQL
Shared-Disk Architecture

- Only shared disks
- No contention for memory and high availability
- Typically 1-10 machines

Interconnection Network (SAN + SCSI)
Shared-Nothing Architecture*

- Uses cheap, commodity hardware
- No contention for memory and high availability
- Theoretically can scale infinitely
- Hardest to implement on

Interconnection Network (TCP)

P P P

M M M

D D D

teradata.  
Apache Spark

MySQL™ Cluster
Main tradeoff is administration difficulty vs ability to scale

Shared-Memory Architecture
- Easy to work on

Shared-Disk Architecture
- Easy to scale

Shared-Nothing Architecture*

If you can’t scale, your product dies, and everyone loses their job
Implementations for Database Parallelism

- Architecture Parallelism
  - Shared Memory
  - Shared Disk
  - Shared Nothing*

- Query Parallelism
  - Inter-Query Parallelism
  - Intra-Query Parallelism
    - Inter-Operator Parallelism
    - Intra-Operator Parallelism*
Inter-Query Parallelism

- Each transaction is processed on a separate node
- Scales very well for **lots of simple transactions**
Inter-Operator Parallelism

- Each operator is processed on a separate node
- Scales very well for complex analytical queries
Intra-Operator Parallelism*

- Each operator is processed by multiple nodes
- Scales well in general
From here, we will assume a system that consists of multiple commodity machines on a common network where nodes may carry out specified relational operations.

New problem: *Where does the data go?*
Unpartitioned Table

- Simplest choice if data can fit on a single node
- Might result in being a bottleneck
Block Partitioning

Tuples are horizontally partitioned arbitrarily in equally sized blocks

\[ B(R) = K \]

\[ B(R_1) = K/N \]

\[ B(R_2) = K/N \]

\[ B(R_N) = K/N \]
Node contains tuples partitioned by hash on chosen attributes

\[ R_1, 1 = h(A) \% N \]
\[ R_2, 2 = h(A) \% N \]
\[ R_N, 0 = h(A) \% N \]
Node contains tuples in chosen attribute ranges

\[ A \]

\[ R_1, -\infty < A \leq v_1 \]

\[ R_2, v_1 < A \leq v_2 \]

\[ R_N, v_N < A < \infty \]

N nodes
Hashing data to nodes is very good when the attribute chosen better approximates a uniform distribution.

Keep in mind: Certain nodes will become bottlenecks if a poorly chosen attribute is hashed.
So how do we get data to the right nodes for our operations?
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:
R is block partitioned

SELECT *
FROM R
GROUP BY R.A
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:
R is block partitioned

\[ \text{SELECT } * \]
\[ \text{FROM R} \]
\[ \text{GROUP BY R.A} \]
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:
R is block partitioned

SELECT *
FROM R
GROUP BY R.A
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:
- $R$ is block partitioned

SQL:
```
SELECT *
FROM R
GROUP BY R.A
```

Diagram:
- Node 1:
  - $A \ldots$
  - 1 \ldots
  - 2 \ldots

- Node 2:
  - $A \ldots$
  - 2 \ldots
  - 3 \ldots

- Node 3:
  - $A \ldots$
  - 3 \ldots
  - 1 \ldots

$\gamma_{R.A}$
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:
- \( R \) is block partitioned

```
SELECT * 
FROM R 
GROUP BY R.A
```
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Would I need to shuffle if R was hash or range partitioned?
Implicit Union

Parallel query plans implicitly union at the end

Output

$\gamma_{R.A}$

Node 1

Node 2

Node 3

hash R.A

hash R.A

hash R.A

Node 1

Node 2

Node 3

A

1

2

3

A

1

2

3

...
Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes
2. Local join

Assume:
R and S are block partitioned

```
SELECT * 
FROM R, S
WHERE R.A = S.A
```

Node 1

Node 2

Node 3

\( \bowtie_{R.A=S.A} \)
Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes
2. Local join

Assume:
R and S are block partitioned

```
SELECT * 
FROM R, S 
WHERE R.A = S.A
```
1. Hash shuffle tuples on join attributes
2. Local join

If $S$ was **hash** partitioned on $A$ (on the same hash function) would I need to shuffle $S$? $R$?
1. Hash shuffle tuples on join attributes
2. Local join

If S was range partitioned on A would I need to shuffle S? R?
Broadcast Join

1. Broadcast unpartitioned table
2. Local join

Assume:
S is unpartitioned and small.

```
SELECT *
FROM R, S
WHERE R.A = S.A
```
Broadcast Join

1. Broadcast unpartitioned table
2. Local join

Assume:
S is unpartitioned and small.

SELECT *
FROM R, S
WHERE R.A = S.A

Doesn’t matter how R is partitioned!

Broadcast all of S
All queries can be parallelized!

```
SELECT R.A
FROM R, S
WHERE R.A = S.A AND R.A > 10
GROUP BY R.A
HAVING MAX(S.B) < 10
```
Parallel Query Plan Example

Assume:
R is block partitioned
S is hash partitioned on A

\[
\begin{align*}
\pi_{R.A} \\
\sigma_{\max(S.B) < 10} \\
\gamma_{R.A, \max(S.B) \rightarrow \max(S.B)} \\
\bowtie_{R.A = S.A} \\
\sigma_{R.A > 10} \\
R & \quad S
\end{align*}
\]
Parallel Query Plan Example

Assume:
R is block partitioned
S is hash partitioned on A

\[ \pi_{R.A} \]

\[ \sigma_{\text{maxSB}<10} \]

\[ \gamma_{R.A, \text{max}(S.B) \rightarrow \text{maxSB}} \]

\[ \bowtie_{R.A = S.A} \]

\[ \sigma_{R.A > 10} \]

\[ \sigma_{R.A > 10} \]

\[ \sigma_{R.A > 10} \]

Node 1

Node 2

Node 3
Parallel Query Plan Example

Assume:
R is block partitioned
S is hash partitioned on A
Parallel Query Plan Example

Assume:
- R is block partitioned
- S is hash partitioned on A

The diagram shows the query plan for the join operation. The nodes are labeled with operations:
- \( \pi_{R.A} \) for projection on A
- \( \sigma_{maxSB<10} \) for selection on \( SB < 10 \)
- \( \gamma_{R.A, max(S.B) \rightarrow maxSB} \) for join on \( R.A = S.A \)
- \( \sigma_{R.A>10} \) for selection on \( R.A > 10 \)
- hash R.A for hash partitioning on A

The nodes are connected to represent the parallel execution plan:
- Node 1: \( \pi_{R.A} \sigma_{maxSB<10} \gamma_{R.A, max(S.B) \rightarrow maxSB} \)
- Node 2: \( \pi_{R.A} \sigma_{maxSB<10} \gamma_{R.A, max(S.B) \rightarrow maxSB} \)
- Node 3: \( \pi_{R.A} \sigma_{maxSB<10} \gamma_{R.A, max(S.B) \rightarrow maxSB} \)
Next Time

- Programming with the Java Spark API