Parallel DBs
Why Scale?

Scan of 1 PB at 300MB/s (SATA r2 Limit)
Why Scale Up?

Scan of 1 PB at 300MB/s (SATA r2 Limit)

~1 Hour
Why Scale Up?

Scan of 1 PB at 300MB/s (SATA r2 Limit)

~1 Hour → ~3.5 Seconds

(x1000)
Data Parallelism

Replication

Partitioning
Operator Parallelism

• Pipeline Parallelism: A task breaks down into stages; each machine processes one stage.

• Partition Parallelism: Many machines doing the same thing to different pieces of data.
Types of Parallelism

- Both types of parallelism are natural in a database management system.

```sql
SELECT SUM(...) FROM Table WHERE ...
```
DBMSes: The First || Success Story

• Every major DBMS vendor has a || version.

• Reasons for success:
  • Bulk Processing (Partition ||-ism).
  • Natural Pipelining in RA plan.
  • Users don’t need to think in ||.
Types of Speedup

- **Speed-up ||-ism**
  - More resources = proportionally less time spent.

- **Scale-up ||-ism**
  - More resources = proportionally more data processed.
Parallelism Models

- CPU
- Memory
- Disk
Parallelism Models

How do the nodes communicate?
Parallelism Models

Option 1: “Shared Memory” available to all CPUs

e.g., a Multi-Core/Multi-CPU System
Parallelism Models

Option 2: Non-Uniform Memory Access.

Used by most AMD servers
Parallelism Models

Option 3: “Shared Disk” available to all CPUs

Each node interacts with a “disk” on the network.
Parallelism Models

Option 4: “Shared Nothing” in which all communication is explicit.

Examples include MPP, Map/Reduce. Often used as basis for other abstractions.
Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates
Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates
Parallelism & Distribution

- **Distribute** the Data
  - Redundancy
  - Faster access

- **Parallelize** the Computation
  - Scale up (compute faster)
  - Scale out (bigger data)
Operator Parallelism

- **General Concept**: Break task into individual units of computation.

- **Challenge**: How much data does each unit of computation need?

- **Challenge**: How much data transfer is needed to allow the unit of computation?

Same challenges arise in Multicore, CUDA programming.
Parallel Data Flow

No Parallelism
Parallel Data Flow

N-Way Parallelism
Parallel Data Flow

Chaining Parallel Operators
Parallel Data Flow

One-to-One Data Flow (”Map”)
Parallel Data Flow

\[ B_1 \rightarrow A_1 \rightarrow B_N \rightarrow A_N \]

One-to-One Data Flow
Parallel Data Flow

**Extreme 1**
*All-to-All*
All nodes send all records to all downstream nodes

**Extreme 2**
*Partition*
Each record goes to exactly one downstream node

Many-to-Many Data Flow
Parallel Data Flow

Many-to-One Data Flow ("Reduce/Fold")
Parallel Operators

| Select | Project | Union (bag) |

What is a logical “unit of computation”? (1 tuple)

Is there a data dependency between units? (no)
Parallel Aggregates

**Algebraic**: Bounded-size intermediate state (Sum, Count, Avg, Min, Max)

**Holistic**: Unbounded-size intermediate state (Median, Mode/Top-K Count, Count-Distinct; Not Distribution-Friendly)
Fan-In Aggregation
Fan-In Aggregation

SUM

8 Messages

A₁ A₂ A₃ A₄ A₅ A₆ A₇ A₈
Fan-In Aggregation

2 Messages (each)

4 Messages

SUM

SUM_1
A_1
A_2

SUM_2
A_3
A_4

SUM_3
A_5
A_6

SUM_4
A_7
A_8
Fan-In Aggregation

2 Messages (each)

SUM

SUM'

SUM

SUM'

SUM

SUM'

SUM

SUM

A1 A2 A3 A4 A5 A6 A7 A8

2 Messages

2 Messages
Fan-In Aggregation

If Each Node Performs K Units of Work...
(K Messages)
How Many Rounds of Computation Are Needed?

\[ \log_K(N) \]
Fan-In Aggregation Components

\[ \text{Combine(Intermediate}_1, \ldots, \text{Intermediate}_N) = \text{Intermediate} \]
\[ <\text{SUM}_1, \text{COUNT}_1> \otimes \ldots \otimes <\text{SUM}_N, \text{COUNT}_N> \]
\[ = <\text{SUM}_1 + \ldots + \text{SUM}_N, \text{COUNT}_1 + \ldots + \text{COUNT}_N> \]

\[ \text{Compute(Intermediate)} = \text{Aggregate} \]
\[ \text{Compute}(<\text{SUM}, \text{COUNT}>) = \frac{\text{SUM}}{\text{COUNT}} \]
Parallel Joins

FOR i IN 1 to N
FOR j IN 1 to K
JOIN(Block i of R, Block j of S)

One Unit of Computation
Parallel Joins

K Partitions of S

<table>
<thead>
<tr>
<th>N Partitions of R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1 of R</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>Block 1 of S</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Block N of R</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>Block 1 of S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>K Partitions of S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 1 of R</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>Block K of S</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Block N of R</td>
</tr>
<tr>
<td>×</td>
</tr>
<tr>
<td>Block K of S</td>
</tr>
</tbody>
</table>
Parallel Joins

\[ R[1] \bowtie 1 \bowtie R[2] \bowtie \ldots \bowtie R[N] \bowtie S[1] \bowtie S[2] \bowtie \ldots \bowtie S[K] \Rightarrow \text{UNION} \]
Parallel Joins

How much data needs to be transferred?

How many “units of computation” do we create?
Parallel Joins

What if we partitioned “intelligently”? 
Parallel Joins

\[
\begin{array}{cccc}
\text{Hash}(S.B) \mod 4 & 0 & 1 & 2 & 3 \\
\text{Hash}(R.B) \mod 4 & \sqrt & \times & \times & \times \\
0 & \sqrt & \times & \times & \times \\
1 & & \sqrt \\
2 & & & \sqrt \\
3 & & & \sqrt \\
\end{array}
\]

\textbf{R} \bowtie_{B} \textbf{S}: Which Partitions of S Join w/ Bucket 0 of R?
### Parallel Joins

<table>
<thead>
<tr>
<th>S.B</th>
<th>R.B</th>
<th>B&lt;25</th>
<th>25≤B&lt;50</th>
<th>50≤B&lt;75</th>
<th>75≤B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B&lt;25</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>25≤B&lt;50</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>50≤B&lt;75</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>75≤B</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

\[ R \bowtie_{R.B < S.B} S: \] Which Partitions of S Can Produce Output?
Distributing the Work

Let’s start simple… what can we do with no partitions?

R and S may be any RA expression…
Distributing the Work

No Parallelism!
Distributing the Work

All of R and All of S get sent!

Lots of Data Transfer!
Better! We can guess whether R or S is smaller.
Distributing the Work

What can we do if R is partitioned?

\[ U \]

\[ \bowtie_B R_1 \]

\[ S \]

\[ \bowtie_B R_2 \]
Distributing the Work

There are lots of partitioning strategies, but this one is interesting....
Distributing the Work

... it can be used as a model for partitioning S...
Distributing the Work

... it can be used as a model for partitioning S...
Distributing the Work

...and neatly captures the data transfer issue.
Distributing the Work

So let’s use it:

$S_i$ joins with $R_1, R_2, \ldots, R_N$ locally.

Goal: Minimize amount of data sent from $R_k$ to $S_i$

Solution 1: Use a partitioning strategy

Solution 2: “Hints” to figure out what $R_k$ should send
Sending Hints

\( R_k \bowtie_B S_i \)

The naive approach...
Sending Hints

\[ R_k \bowtie_B S_i \]

The naive approach...

Node 1

Node 2

Send me \( R_k \)
Sending Hints

$R_k \bowtie B S_i$

The naive approach...

Node 1

$R_k$

Node 2

$S_i$
Sending Hints

$R_k \bowtie_B S_i$

The smarter approach…
Sending Hints

\[ R_k \bowtie_B S_i \]

The smarter approach...

Node 1

Node 2
Sending Hints

\[ R_k \bowtie_B S_i \]

The smarter approach…

Node 1

\(<1,A>\>
\(<2,B>\>
\(<2,C>\>
\(<3,D>\>
\(<4,E>\>

Node 2

\(<2,X>\>
\(<3,Y>\>
\(<6,Y>\>
Sending Hints

\[ R_k \bowtie_B S_i \]

The smarter approach...

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>

Send me rows with a ‘B’ of 2,3, or 6
Sending Hints

\[ R_k \bowtie_B S_i \]

The smarter approach...

Node 1

\(<1,A>\>
\(<2,B>\>
\(<2,C>\>
\(<3,D>\>
\(<4,E>\>

Node 2

\(<2,X>\>
\(<3,Y>\>
\(<6,Y>\>

Send me rows with a ‘B’ of 2,3, or 6

This is called a semi-join.
Sending Hints

Now Node 1 sends as little data as possible…

… but Node 2 needs to send a lot of data.

Can we do better?
Sending Hints

\[ R_k \bowtie_B S_i \]

Strategy 1: Parity Bits

Node 1

\(<1,A>1\>
\(<2,B>0\>
\(<2,C>0\>
\(<3,D>1\>
\(<4,E>0\>

Node 2

\(0<2,X>\>
\(0<6,Y>\>
Sending Hints

\[ R_k \bowtie_B S_i \]

Strategy 1: Parity Bits

Node 1

- \(<1,A>\)_1
- \(<2,B>\)_0
- \(<2,C>\)_0
- \(<3,D>\)_1
- \(<4,E>\)_0

Node 2

- \(<2,X>\)_0
- \(<6,Y>\)_0

Send me data with a parity bit of ‘0’
Sending Hints

$R_k \bowtie_B S_i$

Strategy 1: Parity Bit

Node 1 sending too much is ok!
(Node 2 still needs to compute $\bowtie_B$)

**Node 1**

- $<1,A> \uparrow_1$
- $<2,B> \downarrow_0$
- $<2,C> \downarrow_0$
- $<3,D> \uparrow_1$
- $<4,E> \downarrow_0$

**Node 2**

Send me data with a parity bit of ‘0’

- $0<2,X> \downarrow_0$
- $0<6,Y> \downarrow_0$

Problem: One parity bit is too little
Sending Hints

\[ R_k \bowtie_B S_i \]

Strategy 2: Parity Bits

Node 1:
- \(<1,A>_{01}\)
- \(<2,B>_{10}\)
- \(<2,C>_{10}\)
- \(<3,D>_{11}\)
- \(<4,E>_{00}\)

Node 2:
- \(10<2,X>\)
- \(11<3,Y>\)
- \(10<6,Y>\)

Problem: Almost as much data as \(\pi_B\)

Send me data with parity bits 10 or 11.
Sending Hints

Can we summarize the parity bits?
Bloom Filters

Alice
Bob
Carol
Dave
Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter
Bloom Filters

Bloom Filter Guarantee
Test **definitely** returns Yes if the element is in the set
Test **usually** returns No if the element is not in the set

Is Alice part of the set? Yes
Is Eve part of the set? No
Is Fred part of the set? Yes
Bloom Filters

A Bloom Filter is a bit vector

M - # of bits in the bit vector
K - # of hash functions

For ONE key (or record):
For i between 0 and K:
    bitvector[ hashi(key) % M ] = 1

Each bit vector has ~K bits set
Bloom Filters

Filters are combined by Bitwise-OR

\[ (\text{Key 1} \mid \text{Key 2}) = 01111110 \]

How do we test for inclusion?

\[ (\text{Key} \& \text{Filter}) == \text{Key} \]

\[
\begin{align*}
(\text{Key 1} \& S) &= 00101010 \quad \checkmark \\
(\text{Key 3} \& S) &= 00000110 \quad \times \quad \text{False Positive} \\
(\text{Key 4} \& S) &= 01001100 \quad \checkmark 
\end{align*}
\]
Sending Hints

$R_k \bowtie_B S_i$

Strategy 3: Bloom Filters

Node 1

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

Node 2

<2,X>
<3,Y>
<6,Y>
Sending Hints

\[ R_k \bowtie B \ S_i \]

Strategy 3: Bloom Filters

Node 1

\(<1,A>\>
\(<2,B>\>
\(<2,C>\>
\(<3,D>\>
\(<4,E>\>

Node 2

\(<2,X>\>
\(<3,Y>\>
\(<6,Y>\>

Send me rows with a ‘B’ in the bloom filter summarizing the set \{2,3,6\}
Sending Hints

\[ R_k \bowtie_B S_i \]

Strategy 3: Bloom Filters

This is called a **bloom-join**.

**Node 1**

\(<1,A>\>
\(<2,B>\>
\(<2,C>\>
\(<3,D>\>
\(<4,E>\>

**Node 2**

\(<2,X>\>
\(<3,Y>\>
\(<6,Y>\>

Send me rows with a ‘B’ in the bloom filter summarizing the set \(\{2,3,6\}\).
Bloom Filters

Probability that 1 bit is set by 1 hash fn

\[ \frac{1}{m} \]
Bloom Filters

Probability that 1 bit is not set by 1 hash fn

$1 - \frac{1}{m}$
Bloom Filters

Probability that 1 bit is not set by k hash fns

\[(1 - \frac{1}{m})^k\]
Bloom Filters

Probability that 1 bit is not set by $k$ hash fns for $n$ records

$$(1 - 1/m)^{kn}$$

So for an arbitrary record, what is the probability that all of its bits will be set?
Bloom Filters

Probability that 1 bit is set by $k$ hash functions for $n$ records

$$1 - (1 - \frac{1}{m})^{kn}$$
Bloom Filters

Probability that all k bits are set by k hash functions for n records

\[ \approx (1 - (1 - 1/m)^{kn})^k \]

\[ \approx (1 - e^{-kn/m})^k \]
Bloom Filters

Minimal $P[\text{collision}]$ is at $k \approx c \cdot m/n$
Bloom Filters

\[ k \approx c \cdot \frac{m}{n} \]

\[ \frac{m}{k} \approx cn \]

m is linearly related to n (for a fixed k)
Bloom Join

- Node 2 Computes Bloom Filter for Local Records

- Node 2 Sends Bloom Filter to Node 1

- Node 1 Matches Local Records Against Bloom Filter

- Node 1 Sends Matched Records to Node 2
  - Superset of “useful” records

- Node 2 Performs Join Locally