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

\( \times 1000 \)
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.

```
SELECT SUM(...) FROM Table WHERE ...
```

LOAD  SELECT  AGG  Combine
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 \(\parallel\)-ism**
  - More resources = proportionally less time spent.

- **Scale-up \(\parallel\)-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

- CPU
- Memory
- Disk

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

$B_1 \cdots \cdots \cdots \cdots \cdots B_N$

$A_1 \cdots \cdots \cdots \cdots \cdots A_N$

Chaining Parallel Operators
Parallel Data Flow

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

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 Operators

A_1 \rightarrow \ldots \rightarrow A_N

Select | Project | Union (bag)

1/N Tuples | \ldots | 1/N Tuples
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

N Partitions of R

Block 1 of R \(\times\) Block 1 of S

Block 1 of R \(\times\) Block K of S

Block N of R \(\times\) Block 1 of S

Block N of R \(\times\) Block K of S

Z

K

Z
Parallel Joins


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**

<table>
<thead>
<tr>
<th>Hash(R.B)%4</th>
<th>Hash(S.B)%4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><img src="hash0" alt="√" /></td>
<td><img src="hash0" alt="√" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="hash1" alt="√" /></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**R \bowtie B S**: Which Partitions of S Join w/ Bucket 0 of R?
Parallel Joins

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

**R \bowtie_{R.B < S.B} S:** Which Partitions of S Can Produce Output?
Parallel Joins

Use partitioning to eliminate units of computation

Exactly the same idea as External Hash Join (Called Theta Join for Inequalities)
Bloom Join

<table>
<thead>
<tr>
<th>No Specific Partitioning</th>
<th>No Specific Partitioning</th>
<th>No Specific Partitioning</th>
<th>No Specific Partitioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>

What if the join is highly selective… Can we detect which tuples are useful?
Bloom Join

**Goal:** Summarize which tuples are useful for the join?

False positives: OK
False negatives: NOT OK
Bloom Join

**Strategy 1**: Parity Bit

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td></td>
</tr>
<tr>
<td>&lt;1,...&gt; 1</td>
<td></td>
</tr>
<tr>
<td>&lt;2,...&gt; 0</td>
<td></td>
</tr>
<tr>
<td>&lt;3,...&gt; 1</td>
<td></td>
</tr>
<tr>
<td>&lt;4,...&gt; 0</td>
<td></td>
</tr>
</tbody>
</table>

Send me data w/ parity bit 0
Bloom Join

**Strategy 1**: Parity Bit

Node 1

- `<1,…>`: 1
- `<2,…>`: 0
- `<3,…>`: 1
- `<4,…>`: 0

Node 2

- `<2,…>`: 0
- `<4,…>`: 1
- `<3,…>`: 1

Send me data w/ parity bit 0 or 1
**Bloom Join**

**Strategy 2: Multiple Parity Bits**

Node 1

```
<1,…) 01
<2,…) 10
<3,…) 11
<4,…) 00
```

Node 2

```
10 <2,…)  
11 <3,…)  
```

Send me data w/ parity bits 10, 11

What’s the problem with this?
A Simplified Bloom Join

Key 1 00101010
Key 2 01010110
Key 3 10000110
Key 4 01001100

How do we summarize?

Bitwise OR

e.g. (Key 1 | Key 2) = 01111110

How do we test for inclusion?

(Key & Summary) == Key?

(Key 1 & S) = 00101010
(Key 3 & S) = 00000110
(Key 4 & S) = 01001100

False Positive
Bloom Filters

Generating a bit vector for a key:

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

For ONE key/record:
For i between 0 and K:
\[
\text{bitvector}[\ hash_i(key) \mod M \ ] = 1
\]

Each bit vector has \(\sim K\) bits set
Bloom Filters

Probability that 1 bit is set by 1 hash fn

$1/m$
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

$\left( 1 - \frac{1}{m} \right)^k$
Bloom Filters

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

\((1 - \frac{1}{m})^k n\)

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 fns for $n$ records

$$1 - (1 - \frac{1}{m})^{k\cdot n}$$
Bloom Filters

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

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

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

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

\[ k \approx cn \]

\[ m \]

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
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

A_1 \cdots A_N \rightarrow \text{SUM}
Fan-In Aggregation

SUM

8 Messages
Fan-In Aggregation

2 Messages (each)

SUM

SUM_1

A_1

SUM_2

A_2

A_3

SUM_3

A_4

A_5

SUM_4

A_6

A_7

A_8

4 Messages

2 Messages (each)

SUM

56
Fan-In Aggregation

2 Messages

2 Messages (each)
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

Combine(Intermediate$_1$, …, Intermediate$_N$)
   = Intermediate

<SUM$_1$, COUNT$_1$> $\otimes$ ... $\otimes$ <SUM$_N$, COUNT$_N$>
   = <SUM$_1$+...+SUM$_N$, COUNT$_1$+...+COUNT$_N$>

Compute(Intermediate) = Aggregate

Compute(<SUM, COUNT>) = SUM / COUNT