Parallel DBs

April 25, 2017
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 ||-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

- 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

$A_I \cdots A_N$

N-Way Parallelism
Parallel Data Flow

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

<table>
<thead>
<tr>
<th>Select</th>
<th>Project</th>
<th>Union (bag)</th>
</tr>
</thead>
</table>

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

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

Select  Project  Union (bag)

\[ A_i \quad \cdots \quad A_N \]

1/N Tuples  \cdots  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 \( \Join \) Block 1 of S

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

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

Block N of R \( \Join \) Block 1 of S
Practical Concerns

Where does the computation happen?
How does the data get there?
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

Lots of Data Transfer!

All of R and All of S get sent!
Distributing the Work

All of R get sent

Better! We can guess whether R or S is smaller.
Distributing the Work

What can we do if R is partitioned?

U

⋈

R_1

S

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.
### Parallel Joins

**R \( \bowtie \) S:** Which Partitions of S Join with Bucket 0 of R?

<table>
<thead>
<tr>
<th>Hash(R.B)%4</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash(S.B)%4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Parallel Joins

<table>
<thead>
<tr>
<th>S.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>R.B</td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
</tr>
<tr>
<td></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
</tr>
<tr>
<td></td>
<td><a href="#">☓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
</tr>
<tr>
<td></td>
<td><a href="#">☓</a></td>
<td><a href="#">☓</a></td>
<td><a href="#">✓</a></td>
<td><a href="#">✓</a></td>
</tr>
<tr>
<td>R &lt;=_{R.B} &lt; S.B</td>
<td>S:</td>
<td>Which Partitions of S Can Produce Output?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Can we further reduce the amount of data sent?
Sending Hints

$R_k \bowtie_B S_i$

The naive approach...
Sending Hints

\[ R_k \bowtie_B S_i \]

The naive approach...
Sending Hints

\[ R_k \bowtie_B S_i \]

The naive approach...
Sending Hints

\[ R_k \bowtie_B S_i \]

The smarter approach…
Sending Hints

\[ R_k \bowtie B \ S_i \]

The smarter approach…
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…

Nodes:

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

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

**Node 2**
- $<2,X>$
- $<3,Y>$
- $<6,Y>$

Send me rows with a ‘B’ of 2,3, or 6
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

\[ R_k \Join_B S_i \]

**Strategy 1**: Parity Bits

<table>
<thead>
<tr>
<th>Node 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1,A&gt;</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;2,B&gt;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;2,C&gt;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;3,D&gt;</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;4,E&gt;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>&lt;2,X&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>&lt;6,Y&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sending Hints

\[ R_k \otimes_B S_i \]

**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 \(<6,Y>\)

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

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

Problem: **One** parity bit is too little

Node 2

- \(<2,X>\> 0\)
- \(<6,Y>\> 0\)

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

\[ R_k \otimes_B S_i \]

**Strategy 1**: Parity Bit

---

### Node 1

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

### Node 2

- 0 \(<2, X>\)
- 1 \(<3, Y>\)
- 0 \(<6, Y>\)

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**

- \(<2,B>\)
- \(<2,C>\)
- \(<3,D>\)
- Send me data with parity bits 10 or 11
- \(10<2,X>\)
- \(11<3,Y>\)
- \(10<6,Y>\)

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

Can we summarize the parity bits?
Bloom Filters

Alice
Bob
Carol
Dave
Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter
Bloom Filters

Alice
Bob
Carol
Dave

Bloom Filter

Is Alice part of the set?
Yes

Is Eve part of the set?
No

Is Fred part of the set?
Yes

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
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 \( K \):
\[
\text{bitvector}[\text{hash}_i(\text{key}) \mod M] = 1
\]

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

Filters are combined by Bitwise-OR

\[
\text{(Key 1 | Key 2)} \quad \Rightarrow \quad 01111110
\]

How do we test for inclusion?

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

- Key 1 \(00101010\)
- Key 2 \(01010110\)
- Key 3 \(10000110\)
- Key 4 \(01001100\)

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

False Positive
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\}.
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

\[ \text{SUM} \quad A_i \quad A_N \]
Fan-In Aggregation

SUM

8 Messages

A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4 \rightarrow A_5 \rightarrow A_6 \rightarrow A_7 \rightarrow A_8
Fan-In Aggregation

4 Messages

2 Messages (each)

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

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

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

Compute(Intermediate) = Aggregate

Compute($<\text{SUM}, \text{COUNT}>$) = SUM / COUNT