Why Parallel Data Access?

SELECT * FROM mydata

1 Terabyte (= 1024 GB)

10 MB/s

1.2 days to scan

1000 x parallel 10 MB/s

1.5 mins to scan!
Parallel DBMS

• eBay’s main Teradata data warehouse (DW):
  – > 2 petabytes of user data
  – 10s of 1000s of users
  – Millions of queries per day
  – 72 nodes
  – > 140 GB/sec of I/O, or 2 GB/node/sec

• eBay’s Greenplum DW
  – 6 1/2 petabytes of user data
  – 96 nodes
  – 200 MB/node/sec of I/O

• Walmart – 2.5 petabytes

• Bank of America – 1.5 petabytes

• Some parallel DBMSs besides the usual Oracle-IBM-MS trio:
  – Teradata
  – Netezza
  – Vertica
  – DATAllegro
  – Greenplum
  – Aster Data
  – Infobright
  – Kognitio, Kickfire, Dataupia, ParAccel, Exasol, ...

1/14/2013
Lipyeow Lim -- University of Hawaii at Manoa
Parallelism

**Pipeline parallelism**
- many machines each doing one step in a multi-step process.

**Partition parallelism**
- many machines doing the same thing to different pieces of data.

---

**Parallelism is natural to DBMS processing**

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1:step1</td>
<td>Q1:step2</td>
<td>Q1:step3</td>
</tr>
<tr>
<td>Q2:step1</td>
<td>Q2:step2</td>
<td>Q2:step3</td>
</tr>
<tr>
<td>Q3:step1</td>
<td>Q3:step2</td>
<td>Q3:step3</td>
</tr>
<tr>
<td>Q4:step1</td>
<td>Q4:step2</td>
<td></td>
</tr>
<tr>
<td>Q5:step1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing processing of queries across nodes.
Parallelism Terminology

• Speed-up
  – Same job + more resources  
    = less time

• Scale-up
  – Bigger job + more resources  
    = same time

• Transaction scale-up
  – More clients + more resources  
    = same time
Parallel Architecture: Share What?

Shared Memory (SMP)
- Easy to program
- Expensive to build
- Difficult to scaleup

Shared Disk
- Hard to program
- Cheap to build
- Easy to scaleup

Shared Nothing (network)

Sequent, SGI, Sun

VMScluster, Sysplex

Tandem, Teradata, SP2
Different Types of DBMS Parallelism

• Intra-operator parallelism
  – get all machines working to compute a given operation (scan, sort, join)

• Inter-operator parallelism
  – each operator may run concurrently on a different site (exploits pipelining)

• Inter-query parallelism
  – different queries run on different sites

• We’ll focus on intra-query parallelism
Parallel vs Distributed DBMS

• A **parallel** database system
  – Parallelize various operations such as loading data, building indexes, evaluating queries
  – Often **homogeneous**: Every node runs same type of DBMS.

• A **distributed** database system
  – Data is physically stored across several (geographical) sites
  – Distribution governed by factors like local ownership & increased availability
  – Often **heterogeneous**: Different sites run different DBMSs (different RDBMSs or even non-relational DBMSs).

• The boundaries of these traditional definitions are blurring.
Data Partitioning & Fragmentation

• Parallel DB
  – Data partitioning

• Distributed DB
  – Fragmentation

• Same basic problem: How do we break up the data (tables) and spread them amongst the “nodes”
  – Horizontal vs Vertical
  – Range vs Hash
  – Replication

• DB user’s view should be one single table.
Automatic Data Partitioning

Partitioning a table:

**Range**
- Good for equijoins, range queries, group-by

**Hash**
- Good for equijoins

**Round Robin**
- Good to spread load

- Shared disk and memory less sensitive to partitioning,
- Shared nothing benefits from "good" partitioning
Shared-Nothing Architecture

Network

CPU
Memory
Disk

CPU
Memory
Disk

CPU
Memory
Disk

CPU
Memory
Disk
Logical Parallel DBMS Architecture

Network

query → Parallel DB layer

Catalog DB

DBMS

Data Fragments

results
### Horizontal Fragmentation: Range Partition

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>

**Range Partition on rating column**
- Partition 1: $0 \leq \text{rating} < 5$
- Partition 2: $5 \leq \text{rating} \leq 10$

**Partition 1**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

**Partition 2**

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>
Range Partition: Query Processing

- Which partitions?
- Better than non-parallel?

\[
\begin{align*}
\text{SELECT} & \quad \ast \\
\text{FROM} & \quad \text{Sailors S}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \ast \\
\text{FROM} & \quad \text{Sailors S} \\
\text{WHERE} & \quad \text{rating} = 2
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \ast \\
\text{FROM} & \quad \text{Sailors S} \\
\text{WHERE} & \quad \text{age} > 30
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \ast \\
\text{FROM} & \quad \text{Sailors S} \\
\text{WHERE} & \quad \text{rating} < 2 \text{ and } \text{age} < 30
\end{align*}
\]

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>
Horizontal Fragmentation: Hash Partition

- Hash partitioning using hash function
  - Partition = rating \text{mod} 2
Hash Partition: Query Processing

- Which partitions?
- Better than non-parallel?

```
SELECT * 
FROM Sailors S 
```

```
SELECT * 
FROM Sailors S 
WHERE rating = 2 
```

```
SELECT * 
FROM Sailors S 
WHERE age > 30 
```

```
SELECT * 
FROM Sailors S 
WHERE rating < 2 and age < 30 
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>
Vertical Fragmentation/Partition

- Vertical partitioning
  - Use sid as row identifier

```
<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>sid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>23</td>
</tr>
<tr>
<td>58</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>35</td>
</tr>
</tbody>
</table>
```
Vertical Partition: Query Processing

- Which partitions?
- Better than non-parallel?

```
SELECT * FROM Sailors S

SELECT sname FROM Sailors S

SELECT * FROM Sailors S WHERE rating = 2

SELECT sid FROM Sailors S WHERE age > 30

SELECT sid FROM Sailors S WHERE rating < 2 and age < 30
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>4</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>45</td>
</tr>
<tr>
<td>29</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>32</td>
<td>23</td>
</tr>
<tr>
<td>58</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>35</td>
</tr>
</tbody>
</table>
Suppose table is fragmented into 4 partitions on 4 nodes

Replication stores another partition on each node

- What happens when 1 node fails? 2 nodes?
- What happens when a row needs to be updated?
What about joins?

**SELECT** R.sid, R.bid  
**FROM** Sailors S, Reserves R  
**WHERE** S.sid=R.sid AND rating > 8

- Sailors: hash  
  - part = rating mod 2
- Reserves: hash  
  - part = sid mod 2
- Where to perform join?
- What data to ship?

```sql
SELECT R.sid, R.bid
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND rating > 8
```
Distributed Joins

- Consider:
  - Reserves join Sailors
- Depends on:
  - Which node get the query
  - Whether tables are fragmented/partitioned or not

- Node 1 gets query
  - Perform join at Node 3 (or 4) ship results to Node 1 ?
  - Ship tables to Node 1 ?
- Node 3 gets query
  - Fetch sailors in loop ?
  - Cache sailors locally ?
Distributed Joins over Fragments

\[ R \text{ join } S = \sigma_{R.\text{sid}=S.\text{sid}} (R \times S) \]
\[ = \sigma_{R.\text{sid}=S.\text{sid}} ((R1 \cup R2) \times (S1 \cup S2)) \]
\[ = \sigma_{R.\text{sid}=S.\text{sid}} ((R1 \times S1) \cup (R1 \times S2) \cup (R2 \times S1) \cup (R2 \times S2)) \]
\[ = \sigma_{R.\text{sid}=S.\text{sid}} (R1 \times S1) \cup \sigma_{R.\text{sid}=S.\text{sid}} (R1 \times S2) \cup \sigma_{R.\text{sid}=S.\text{sid}} (R2 \times S1) \cup \sigma_{R.\text{sid}=S.\text{sid}} (R2 \times S2) \]
\[ = (R1 \text{ join } S1) \cup (R1 \text{ join } S2) \cup (R2 \text{ join } S1) \cup (R2 \text{ join } S2) \]

This equivalence applies to splitting a relation into pages in a single server DBMS system too!

Equivalent to a union of joins over each pair of fragments
Distributed Nested Loop

• Consider performing R1 join S2 on Node 1

• Page-oriented nested loop join:
  
  For each page r of R1
  Fetch r from local disk
  For each page s of S2
  Fetch s if s∉cache
  Output r join s

  Cost = Npages(R1) * t_d +
        Npages(R1) * Npages(S2) * (t_d + t_s)

• If cache can hold entire S2, cost
  is Npages(R1) * t_d + Npages(S2) * t_s +
  Npages(R1) * Npages(S2) * t_d
Semijoins

- Consider performing R1 join S2 on Node 1
- S2 needs to be shipped to R1
- Does every tuple in S2 join with R1?
- Semijoin:
  - Don’t ship all of S2
  - Ship only those S2 rows that will join with R1
  - Assumes that the join causes a reduction in S2!
- Cost = \(N\text{pages}(R1) \times t_d + N\text{pages}(\pi_{sid}R1) \times t_s + \text{Cost(} \cap \text{)} + N\text{pages}(\sigma_{sid \in jsid} S2) \times t_s + \text{Cost(R1 join } \sigma_{sid \in jsid} S2)\)
Bloomjoins

• Consider performing R1 join S2 on Node 1
• Can we do better than semijoin?
• Bloomjoin:
  – Don’t ship all of \(\pi_{\text{sid}} R1\)
  – Node 1: Ship a “bloom filter” (like a signature) of \(\pi_{\text{sid}} R1\)
    • Hash each sid
    • Set the bit for hash value in a bit vector
    • Send the bit vector v1
  – Node 2:
    • Hash each \(\pi_{\text{sid}} S2\) to bit vector v2
    • Computer \(v1 \cap v2\)
    • Send rows of S2 in the intersection

• False positives
Google Map Reduce
Word Count over a Given Set of Web Pages

see bob throw

see 1
bob 1
throw 1

see spot run

see 1
spot 1
run 1

Can we do word count in parallel?
The MapReduce Framework
(pioneered by Google)
Automatic Parallel Execution in MapReduce (Google)

Handles failures automatically, e.g., restarts tasks if a node fails; runs multiples copies of the same task to avoid a slow task slowing down the whole job.
MapReduce in Hadoop (1)

Figure 2-2. MapReduce data flow with a single reduce task
MapReduce in Hadoop (2)

Figure 2-3. MapReduce data flow with multiple reduce tasks
MapReduce in Hadoop (3)

Figure 2-4. MapReduce data flow with no reduce tasks