Query Optimization in Microsoft SQL Server PDW

Stacey Heideloff
CIS 601
SQL Server PDW

- **Microsoft’s Parallel Data Warehouse**
  - Used for big data analytics
  - Massively Parallel Processing System (MPP)
- **Design**
  - Control Node: manages the compute nodes
    - User-interface for queries
    - Determines query execution plan
    - Issues plan steps to compute nodes
  - Compute Nodes: provide data storage and processing
  - Nodes are connected through Infiniband / Ethernet
SQL Server PDW

- Sold as an appliance
  - Hardware and software bundled together

- Benefits
  - Speed
  - Scalability
  - Availability
SMP Databases

- Symmetric Multi-Processing System (SMP)
  - Multiple CPUs sharing resources
    - OS
    - Memory
    - I/O Devices
  - Processors and resources are connected by a common bus
  - Reliance on shared resources hurts efficiency and fault-tolerance
  - Main constraint in processing speed: memory
MPP Databases

- Massively Parallel Processing System (MPP)
  - Each processor has own OS and memory
  - Can be
    - “Shared nothing” architecture
    - “Shared disk” architecture
  - Shared nothing architecture
    - Each node has its own separate memory and disk
    - A node processes only the rows it has stored for a table
    - Data for a table can be distributed among nodes or replicated across nodes
SQL Server PDW Architecture

- **MPP with Shared-Nothing Architecture**
  - **Control Node**
    - PDW engine
      - Creates parallel query plans
      - Coordinates execution on compute nodes
    - SQL Server Instance
      - Shell database: metadata for distributed database
  - Data Movement Service (DMS)

- **Compute Nodes**
  - SQL Server Instance
    - Stores user data
      - Tables are distributed or replicated
  - Data Movement Service (DMS)
Distributed Tables

Each table row belongs in one distribution on one compute node. This is calculated by hashing the value contained within the defined distribution column for that row.

Hash Function

Row

Compute Node 1
Dist A
Dist B
Dist C
Dist D
Dist E
Dist F
Dist G
Dist H

Compute Node 2
Dist A
Dist B
Dist C
Dist D
Dist E
Dist F
Dist G
Dist H

Compute Node 3
Dist A
Dist B
Dist C
Dist D
Dist E
Dist F
Dist G
Dist H

Compute Node 4
Dist A
Dist B
Dist C
Dist D
Dist E
Dist F
Dist G
Dist H
Replicated Tables

Each table row is copied to all compute nodes.

Row

Broadcast

Compute Node 1
- Dist A
- Dist B
- Dist C
- Dist D
- Dist E
- Dist F
- Dist G
- Dist H

Compute Node 2
- Dist A
- Dist B
- Dist C
- Dist D
- Dist E
- Dist F
- Dist G
- Dist H

Compute Node 3
- Dist A
- Dist B
- Dist C
- Dist D
- Dist E
- Dist F
- Dist G
- Dist H

Compute Node 4
- Dist A
- Dist B
- Dist C
- Dist D
- Dist E
- Dist F
- Dist G
- Dist H
Parallel Query Operation Types

- **SQL Operations**
  - Performed on compute nodes against their DBMS instances
  - Examples
    - Join
    - Select

- **DMS Operations**
  - Transfer data between different nodes
  - Examples
    - Shuffle Move
    - Partition Move
DMS Operations

- **Shuffle Move**
  - Moves rows from each compute node to target tables based on hash value for specified column

- **Partition Move**
  - Moves rows from each compute node to a table on one target node

- **Control-Node Move**
  - Replicates a table on the control node to all compute nodes

- **Broadcast Move**
  - Moves rows from each compute node to target tables on all compute nodes
DMS Operations

- **Trim Move**
  - Performed on a table that is replicated on all compute nodes
  - Turns the table into a distributed table by hashing at each compute node and only keeping hashed results

- **Replicated Broadcast**
  - Replicates a table that was originally on only one compute node

- **Remote Copy To Single Node**
  - Does a remote copy of either a replicated table or distributed table to just one node
Parallel Query Processing Steps

- User submits query to Control Node
- Control Node
  - Parses query
  - Produces distributed execution plan (DSQL plan)
  - Issues execution steps to compute nodes
- Compute Nodes
  - Process their own portions of the query
  - Send results back to Control Node
- Control Node assembles individual pieces into final result and displays to user
  - From user’s view, all work appears to have been done on the control node
Parallel Query Optimization

- **PDW Query Optimizer inputs**
  - SQL Server Query Optimizer results against “shell database”
    - Shell Database
      - Metadata and statistics about tables in distributed database
      - Does not contain any user data
      - Resides on Control Node
  - Data movement operation information

- **PDW Optimization Steps**
  - Runs SQL Server Query Optimizer against “shell database” (system image of data)
  - Exports compact representation of optimization search space called MEMO
  - Considers each execution alternative with respect to data movement costs
  - Selects least-costly plan
Query Example

- Customer Table: partitioned on c_custkey
- Orders Table: partitioned on o_orderkey

Query

```
SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
AND o_totalprice > 100
```
Query Example

- Customer Table: partitioned on c_custkey
- Orders Table: partitioned on o_orderkey

Query
```
SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
AND o_totalprice > 100
```

Problem!
Orders table is not partitioned on o_custkey
Join is not compatible with storage of data
**Query Example**

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>Iname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carey</td>
</tr>
<tr>
<td>10</td>
<td>Houston</td>
</tr>
<tr>
<td>100</td>
<td>Franklin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>3</td>
</tr>
<tr>
<td>1002</td>
<td>300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>Iname</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Jackson</td>
</tr>
<tr>
<td>20</td>
<td>Moore</td>
</tr>
<tr>
<td>200</td>
<td>Clarkson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>100</td>
</tr>
<tr>
<td>2001</td>
<td>30</td>
</tr>
<tr>
<td>2002</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>Iname</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Perry</td>
</tr>
<tr>
<td>30</td>
<td>Roth</td>
</tr>
<tr>
<td>300</td>
<td>Gomez</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>3001</td>
<td>20</td>
</tr>
<tr>
<td>3002</td>
<td>10</td>
</tr>
</tbody>
</table>

**Query:**

```sql
SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
```
**Query Example**

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carey</td>
</tr>
<tr>
<td>10</td>
<td>Houston</td>
</tr>
<tr>
<td>100</td>
<td>Franklin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>3</td>
</tr>
<tr>
<td>1002</td>
<td>300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Jackson</td>
</tr>
<tr>
<td>20</td>
<td>Moore</td>
</tr>
<tr>
<td>200</td>
<td>Clarkson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>100</td>
</tr>
<tr>
<td>2001</td>
<td>30</td>
</tr>
<tr>
<td>2002</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_custkey</th>
<th>lname</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Perry</td>
</tr>
<tr>
<td>30</td>
<td>Roth</td>
</tr>
<tr>
<td>300</td>
<td>Gomez</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_custkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>1</td>
</tr>
<tr>
<td>3001</td>
<td>20</td>
</tr>
<tr>
<td>3002</td>
<td>10</td>
</tr>
</tbody>
</table>

Can’t perform join within Node
Matching data is on other Nodes

```
SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
```
DMS Operation: Re-Hash Orders Table

Now join is possible
Each node performs join on its own data

SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
DSQL Plan

- Data Movement Operation
  - Repartition data in Orders table on o_custkey
    1) Control node broadcasts DMS operation to each compute node
    2) Each compute node begins local execution of the broadcasted DMS operation
      ```sql
      SELECT o_custkey, o_orderdate
      FROM Orders
      WHERE o_totalprice > 100
      ```
      3) Result tuples are hashed on o_custkey, then routed to the appropriate compute node’s temp table

- SQL Operation
  - Control node issues SQL statement on each compute node to locally perform join
    ```sql
    SELECT c_custkey, temp_o_orderdate
    FROM Customer c, Temp_Table tmp
    WHERE c_custkey = tmp_o_custkey
    ```
  - Control node reads result tuples from each compute node, assembles into final result

Original Query
```sql
SELECT c_custkey, o_orderdate
FROM Customer, Orders
WHERE c_custkey = o_custkey
AND o_totalprice > 100
```
Optimization Process

- **PDW Parser**
  - Creates abstract syntax tree (AST) to confirm query uses correct PDW syntax

- **SQL Server Compilation**
  - SQL Server Optimizer is run for query against the shell database
  - Produces MEMO data structure, which holds alternative query plans
  - Selects the optimal serial plan (not the best distributed plan)

- **XML Generator**
  - Takes MEMO data structure and encodes it as XML

- **PDW Query Optimizer**
  - Takes the XML output and adds in data movement strategies
  - Performs cost analysis of these alternative plans
  - Determines best distributed plan using bottom-up optimization
SQL Server Compilation

- Query is transformed into logical query tree
- Initial Memo is created
  - Memo will hold all alternative plans for query
  - Logical query tree is used as initial plan
- Relational algebra rules are applied to entries in Memo to produce logically equivalent plans
  - Ex: generate equivalent join orders
Estimates size of intermediate results for execution alternatives
- Size of base table
- Statistics on columns

Adds physical operator choices to Memo plans
- Determines cost
- Eliminates options that do not meet established bounds

Chooses optimal execution plan
- This is only the best serial plan
Memo Structure

Group: represents all equivalent operator trees producing the same output
Group Expression: an operator that has “Groups” as children
Group 1

- All equivalent expressions that return the content of “Customer” table
  - Logical: Get C
  - Physical: Table Scan, Sorted Index Scan
Memo Structure

Group 4
- All equivalent expressions that join “Customer” and “Orders”
- GroupExpression 4.1
  - All operator trees that have “Join” as root, “Group 1” as first child, and “Group 3” as second child
Group 4

- All equivalent expressions that join “Customer” and “Orders”
- GroupExpression 4.6
  - Hash join operator, “4th GroupExpression in Group 1” as first child, and “3rd GroupExpression in Group 2” as second child
PDW Query Optimizer

- XML Generator takes Memo from SQL Server QO and turns into XML format
- PDW QO parses this XML Memo
  - Creates own Memo structure that can hold PDW operations
- Adds data movement operation choices
- Determines cost of alternative plans
- Chooses optimal execution plan
  - This is best distributed plan
- Creates final DSQSL plan
Cost Model for Distributed Queries

- Data movement time is the major factor in overall query execution time
  - Materializing temp tables

- Cost Model Assumptions
  - DSQL steps execute sequentially
  - No pipelining of DSQL steps
    - Intermediate results always materialized
  - Query assumed to run in isolation
  - All nodes have same hardware
  - Data distributed uniformly across nodes
DMS Operator Costs

Sending Side
- Creader: read tuples from executed SQL query and pack into buffer
- Cnetwork: send buffer over network
- Data is sent asynchronously
- Csource = max(Creader, Cnetwork)

Receiving Side
- Cwriter: unpack tuples from received buffer, prepare for insertion into temp table
- CSQLBulkCopy: bulk insert into temp table
- Bulk insert is done asynchronously
- Ctarget = max(Cwriter, CSQLBulkCopy)

DMS Operation Cost
- CDMS = max(Csource, Ctarget)
Cost of Individual Component

- $C_x = B \times \lambda$
  - $\lambda$: cost per byte
    - Calculated through targeted performance tests
      - “Cost calibration”
    - Is considered constant regardless of number of rows / number of columns / column type
    - Only exception is Creader to account for hashing overhead (Shuffle, Trim)
      - $\lambda_{hash}$
      - $\lambda_{direct}$
  - $B$: number of raw bytes
Cost of Individual Component

- \( C_x = B \times \lambda \)
  - \( B \): number of raw bytes
  - Depends on distribution of input and output streams
    - Distributed Data Streams
      \[ B = \frac{Y \times w}{N} \]
    - Replicated Data Streams
      \[ B = (Y \times w) \]
  - \( Y \): global cardinality
  - \( w \): width of row
  - \( N \): number of nodes
DSQL Generation

- Once plan is determined, DSQl must be generated
- **DSQL Generation Process**
  - PDW Optimizer output is physical operator tree
  - Physical tree is converted into RelOp tree
  - RelOp tree is converted into a PIMOD AST by Qrel library
  - PIMOD script generator turns AST into T-SQL string
- PDW will send SQL statements to compute nodes for execution
- DMS Operations are used to transfer data between nodes
- Results are delivered to Control Node and displayed to user
References