Query Optimization in Microsoft SQL Server PDW

The article done by: Srinath Shankar, Rimma Nehme, Josep Aguilar-Saborit, Andrew Chung, Mostafa Elhemali, Alan Halverson, Eric Robinson, Mahadevan Sankara Subramanian, David DeWitt, César Galindo-Legaria

Microsoft Corporation
SMP and MPP

- Symmetric Multi-Processing (SMP): Having Multiple Processors which share single copy of OS.
- Massively Parallel Processing (MPP): systems, i.e., distributed systems consisting of multiple independent nodes connected by a network.
- MPP: is used to manage and query vast amounts of data.
Microsoft SQL Server Parallel Data Warehouse

- Is a shared-nothing parallel database appliance and is one example of an MPP system.
- Such architecture are especially well suited to data warehouse workloads, where large fact tables can be distributed across nodes.
The control node is responsible for:
- Query parsing
- Creating a DSQL (DSQL operation (SQL&DMS))
- Issuing plan steps to the compute nodes.
- Tracking the execution steps of the plan.
- Assembling the individual pieces of the final results.

Compute nodes provide:
- Data storage
- Query processing

Figure 1: Microsoft SQL Server PDW.
Query Optimization in PDW

- They store the metadata of the distributed tables in a “shell database” on a single SQL Server instance. The “shell database” provides the “single system image” of the data in the appliance.

- Using the shell database, with a given query, we generate the space of execution alternatives (MEMO).

- Using Data movement operations, we make a cost-based decision on the best execution plan for the distributed environment.
MICROSOFT SQL SERVER PDW ARCHITECTURE OVERVIEW

- **Appliance:** The PDW appliance is composed of hardware and software architected to function together as one “box.” Multiple servers are used to implement scale-out query processing in a shared-nothing fashion.

- Allows cost-effective and incremental growth of the appliance by adding external servers or storage

- As the appliance grows over time the server components can be upgraded with more powerful CPU, Memory, storage, etc.

There are two distinct types of nodes that implement the query processing functionality:

1. Control Node.
2. Compute Nodes.
Control Node

- Manages distribution of Query execution across compute nodes
- Accepts client connection to the PDW appliance and manages client authentication
- Manages DMS (Data Movement Service), is communication layer for transferring data between node
- A user can connect to PDW Control Node using variety of client access tools using drivers with connection types, Such as: OLE DB, ODBS, ADO.NET
Compute Nodes

- Each compute node is the host for a single SQL Server instance.
- It also runs a DMS process for communication and data transfer with the other nodes in the appliance.
- Each compute node stores a portion of the user data.
Tables in a PDW appliance can either be:

**Distributed Tables**
- Each table row belongs to one distribution on one compute node. This is calculated by hashing the value contained within the defined distribution column for that row.

**Replicated Table**
- Each table row is copied to all compute nodes.
MICROSOFT SQL SERVER PDW ARCHITECTURE OVERVIEW (cont..)

- **Shell Database**: is a SQL Server database that defines all metadata and statistics about tables, but does not contain any user data.
  - used to store the metadata for the user tables partitioned across the compute nodes.
  - Also store in this database all information regarding users and privileges, so that compilation can check for security and access rights. (no extra cost for PDw)
  - contains global statistics for all the tables in the appliance.

- **Data Movement Service (DMS)**: is responsible for moving data between all the nodes on the appliance. Once instance of DMS runs on each of the control and compute nodes.
  - PDW utilizes temporary tables
  - Some queries that generates no temp table can be streamed from compute node directly back to client.
The DSQL Plan and its Execution

A DSQL plan may include the following types of operations:

- SQL Operations: that are executed directly on the SQL Server DBMS instances on one or more compute nodes.
- DMS Operations: which move data among the nodes in PDW for further processing.
- Temp table operations: that set up staging tables for further processing.
- Return operations: which push data back to the client.
DSQL Plan Example

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

➤ Step1: DMS operation:
➤ repartitions Order Table on O_custkey which is compatible to join

SELECT o_custkey,
o_orderdate
FROM Orders
WHERE o_totalprice > 100
DSQL Plan Example (cont..)

- **Step2: SQL Operation:**
  - that selects tuples for the final result set from each compute node and returns them back to the client.

```sql
SELECT c.c_custkey, 
tmp.o_orderdate 
FROM Customer c, 
Temp_Table tmp 
WHERE c.c_custkey = tmp.o_custkey
```
Cost Based Query Optimization in PDW

Figure 2: Overview of query optimization in PDW.
Figure 3: Parallel query optimization flow: (a) input query, (b) logical query tree, (c) augmented MEMO, (d) best query plan, (e) DSQL plan.
Implementation of PDW QO

- Changes to SQL Server:
  - First change is to support exporting the optimizer search space
  - Second is to extend the query surface to support all constructs of PDW
  - Third is to expand the optimizer search space to include some alternatives that are relevant for distributed query execution
Plan Enumeration

Naïve enumeration (not successful)

A bottom-up optimizer starts by optimizing the smallest expressions in the query, and then uses this information to progressively optimize larger expressions until the optimal physical plan for the full query is found.

```pseudocode
PDWOptimizer()
01 Parse the MEMO XML from SQL server into the PDW MEMO object.
02 Apply MEMO pre-processor rules (bottom-up).
   Example: Fix cardinality estimates of partial aggregates based on PDW topology.
03 Merge equivalent group expressions from the perspective of PDW within the groups (bottom-up).
04 Derive interesting properties of groups (top-down).
   /* ** BOTTOM-UP ENUMERATION ***/
05 For each group in the MEMO, in bottom-up order, do the following:
06 Enumeration step:
   06.1 Enumerate PDW optimization options by considering all possible inputs from child nodes.
      If it's a base group, add a Get operation.
      Apply heuristic pruning.
   06.ii Cost based pruning:
      As PDW group expressions are added, keep only the overall best option
      and the best option per each interesting property.
      Do pruning every time a new PDW group expression is added into the MEMO.
      The maximum cardinality of the set of PDW group expressions in
      a group is (# of interesting properties + 1). //one is added to account for overall
07 Enforcer step
   Add PDW move group expressions based on interesting properties in the current group.
   Apply cost-based pruning similar to step 06.ii above.
   /* ** END BOTTOM-UP ENUMERATION ***/
08 Extract the best overall plan by starting at the best plan in the root group and going
   down the memo to obtain the optimal plan tree.
09 Apply post-optimization rules on the optimal plan tree.
10 Perform DSQL-generation by traversing the optimal plan tree bottom-up.
11 Apply post-DSQL-generation rules on the DSQL plan.
12 Return the DSQL plan to the engine for execution.
```

Figure 4: Pseudo-code for PDW Optimizer (component 4 in Figure 2).
Cost model

- Evaluates the performance of specific plan which includes SQL relational operations and data movement operation

- There is no equivalent to data movement operation inside SQL server, thus we cannot rely on SQL Server optimizer to generate cost for these operation

- **Cost Model Assumptions**: Absence of independent parallelism, Isolation, Homogeneity, etc.
Some of types of Data operator

- **Shuffle move (M-M):** rows are moved from each compute node to targeted table.
- **Partition Move (M-1):** rows are moved from each compute node to targeted table on the target node.
- **Control Node Move:** A table in control is replicated to all compute nodes.
- **Broadcast move:** Rows are moved from each compute node to all targeted table on all compute nodes.
Cost of a DMS Operator

C\text{reader}: \text{Read tuples from the query executed against SQL Server and packing them into a buffer.}

C\text{network}: \text{Send the data buffers over the network}

C\text{writer}: \text{Unpack the tuples from the buffers sent by the sending process, and prepare buffers for insertion into a temporary table.}

C\text{SQLBlkCpy}: \text{Bulk copy operation for insertion of the data buffers into a SQL Server temporary table.}

C\text{source} = \max(C\text{reader}, C\text{network}).

C\text{target} = \max(C\text{writer}, C\text{SQLBlkCpy}).

C\text{DMS} = \max(C\text{source}, C\text{target}).
Costing of an Individual Component

- The number of bytes $B$ to be processed by each individual cost component depends on the distribution properties of the input and output streams.
- $Y$ denote global cardinality
- $w$ – the width of the row
- $N$- denote the number of nodes in the appliance
- $B=(Y\times W)/N$ for distributed data streams.
- $B=Y\times W$ for replicate data stream
DSQL Generation

Figure 6: DSQL Generation.
Figure 7: Parallel query plan for TPCH query 2a
Conclusion

- Use of technology developed for SQL server, the PDW QO goes beyond simple predicate pushing and join reordering
- The cost model of PDW QO is specially crafted to reflect the distributed environment
- Another important aspect of technology reuse was that it shortened the time to build a cost-based optimizer for PDW
- The quality and effectiveness of the result validate the approach
REFERENCES