Query Optimization in Cloud Environment

Cindy Chen
Computer Science Department
University of Massachusetts Lowell
May 31, 2014
OUTLINE

• Introduction
• Our Approach
• Performance Evaluation
• Conclusion and Future Work
Background: Cloud Computing

• Cloud computing is a technology through which service providers provide resources to users online and on demand.

• It has attracted much attention from consumers by providing,
  – Dynamically scalable resources
  – No hassle of investment and maintenance
  – Cost effective solutions.

• Cloud provides an environment where the user can perform tasks as if the data is stored locally when it is actually stored in remote systems.

• Providing such an environment needs a powerful computing, fast execution strategies for tasks and high-speed communication network.
Background: Data Warehouse

Data Source 1 → Data Warehouse → Data Source 2 → Node 1 → Node 2 → Node n → Data Source n → OLAP App 1 → OLAP App 2 → OLAP App n
Background: OnLine Analytical Processing (OLAP)

- Data is more often read than updated
- It enables users to analyze multidimensional data from multiple perspectives
- Mainly used by analytical applications
  - Report generation for sales, marketing, budgeting, forecasting or financial
  - Medical: Analysis of diseases (data mining using historical data)
  - Business Process Management (BPM), decision making for the business growth
- Long interval of read-only queries with batch upload
Background: OLAP Continued..

- Data is usually organized using star schema.
- Star schema consists of one or more fact tables, referencing several dimension tables as shown in the figure.
  - A fact table represents events such as sales transaction. Ex: day, time, quantity, customer-id
  - A dimension table represents descriptive data.

**Product_table**
- product-id
- name
- category
- size
- manufacturer

**Sales_table**
- sale-id
- store-id
- product-id
- customer-id
- date
- quantity

**Store_table**
- store-id
- city
- state

**Customer_table**
- customer-id
- name
- contact
- city
- state

**STAR SCHEMA**
Motivation

• According to Oracle’s survey, 36% Data Warehouse users have performance problems
  – Running reports that involve complex table joins and aggregation.
  – Loading large data volumes into data warehouse and its maintenance
  – Poor metadata scalability

Motivation Continued..

- OLAP applications are read intensive and needs fast response
- Multi-join queries are more common in OLAP
- Rapid growth of data
- Need for a dynamically scalable environment like Cloud
- Less interference on performance along with the growth in data and computing environment
Data Storage Structure

• Storage can be
  – Row oriented
    • Microsoft
    • Oracle
    • Microstrategy
  – Column oriented
    • Vertica
    • Greenplum
    • Sybase
  – Hybrid
    • SAP
Join Operation

• Join operation is used to retrieve data from two or more tables stored in the database.

• Example: Sales \(\bowtie\) Store

• Tables in a database are often related to each other with keys (primary key / foreign key)

• There are different types of joins such as natural join (or inner join), theta join, left outer join and right outer join
Join Operation Continued..

• Find all stores and its location where customer c1 purchased product p2

```
SELECT st.store-id, st.city
FROM sales sa, store st
WHERE sa.store-id = st.store-id
    and sa.customer-id = 'c1'
    and sa.product-id = 'p2'
```
Join Operation Continued..

- Now if the data is distributed into two or more nodes (say 3 nodes – node1, node2 and node3)

<table>
<thead>
<tr>
<th>sale id</th>
<th>store id</th>
<th>product id</th>
<th>customer id</th>
<th>date</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>s1</td>
<td>p1</td>
<td>c2</td>
<td>01/21/2013</td>
<td>1</td>
</tr>
<tr>
<td>0002</td>
<td>s2</td>
<td>p2</td>
<td>c1</td>
<td>01/23/2013</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>store id</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>boston</td>
<td>MA</td>
</tr>
<tr>
<td>s2</td>
<td>lowell</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>nashua</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sale id</th>
<th>store id</th>
<th>product id</th>
<th>customer id</th>
<th>date</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0003</td>
<td>s2</td>
<td>p3</td>
<td>c3</td>
<td>02/12/2013</td>
<td>6</td>
</tr>
<tr>
<td>0004</td>
<td>s3</td>
<td>p1</td>
<td>c1</td>
<td>02/20/2013</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>store id</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>lowell</td>
<td>MA</td>
</tr>
<tr>
<td>s1</td>
<td>boston</td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>nashua</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sale id</th>
<th>store id</th>
<th>product id</th>
<th>customer id</th>
<th>date</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0005</td>
<td>s3</td>
<td>p2</td>
<td>c1</td>
<td>02/24/2013</td>
<td>11</td>
</tr>
<tr>
<td>0006</td>
<td>s3</td>
<td>p3</td>
<td>c3</td>
<td>03/01/2013</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>store id</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3</td>
<td>nashua</td>
<td>NH</td>
</tr>
<tr>
<td>s1</td>
<td>boston</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>lowell</td>
<td></td>
</tr>
</tbody>
</table>

12
Two new nodes are added to the cluster
Number of nodes: 6
Number of communication: 30

Number of nodes: 21
Number of communication: 420
Problem

- Growing number of nodes increases communication during Join operation

- More communication between nodes causes,
  - Network congestion
  - Communication delay
  - Message size overhead due to header processing
  - Affects query performance
Possible Solution

• Reduce the number of communication between nodes
  – Reduces the network congestion and communication delay
    • This can be done by making each node execute query as independently as possible

• Reduce the message size
  – Reduces the message overhead
Our approach

• Storage Structure: stores information of primary key and its associated foreign keys of each table in the database.
  – PK-map
  – Tuple-index-map

• Query processing algorithm: each node processes multi-join query as independently as possible.
It is a schema of an industry which must manage, sell and distribute its products worldwide.
Learning with Purpose

• **PK-map** will have,
  – 1 column for primary key attribute
  – 1 column for each of the foreign key referencing this primary key as shown in Table 1 and Table 2.

• Primary key column stores the primary key attribute value and the foreign key column stores the logical record-id.

• Since all the columns of the map are sorted in its order, compression is applied on foreign key logical record-ids to reduce the size of the map.

  # of records in PK-map = # of records in the dimension table
• **Tuple-index-map** (Table 3) stores the mapping between the logical and actual record-id of the foreign key in the PK-map.

• For the TPC-H schema showed in Figure 1, 7 PK-maps and 8 tuple-index-maps are generated.

<table>
<thead>
<tr>
<th>r_regionkey</th>
<th>n_rk_mindex</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000</td>
<td>0</td>
</tr>
<tr>
<td>00000000001</td>
<td>5</td>
</tr>
<tr>
<td>00000000002</td>
<td>10</td>
</tr>
<tr>
<td>00000000003</td>
<td>15</td>
</tr>
<tr>
<td>00000000004</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 1: RegionKey-map

<table>
<thead>
<tr>
<th>n_rk_mindex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>...........</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>......</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>.....</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

Table 3: RegionKey-tuple-index-map
Reference Graph

- **Rectangle box** represents a table of the star-schema.

- **Directed arrows** connecting these boxes represent the relationship between the tables. E.g. arrow connecting region table to nation table means the primary key of region table is referenced as foreign key in the nation table.

- **d** gives the depth of the table. i.e., Number of other tables linked to it starting from the table with no reference (d=0).
Figure 2: Reference Graph for Star-Schema of Figure 1
Algorithm 1 Query Processing Algorithm

**Input:** query $Q$, reference graph $G$

**Output:** Result of query $Q$

1: Let $T$ be an array of tables referenced in Query

2: Sort $T$ based on $d$ value of $G$ // $d$ value is shown in Fig. 2

3: for each table $t \in T$ do

4: if there is a predicate on non-PK / non-FK then

5: if $d == 0$ for $t$ then

6: Apply predicate on $t$ to get the record ids

7: Store the record-id mapping in the format

8: (rec-id$_1$, rec-id$_2$,.....)

9: Communicate if necessary with other nodes

10: else if any table $t_j$ with $d_j <= d$ referenced by $t$ then

11: Apply predicate on $t$

12: Update the mapping with rec-ids of $t$

13: Perform line 9

14: Eliminate mapping which has no match for $t$

15: else

16: Perform similar to line 6, 9, 14

17: end if

18: else if there is a predicate on PK or FK then

19: if $d == 0$ for $t$ then

20: Scan PK-map and tuple-index-map

21: Perform line 6 to 8

22: else

23: Scan PK-map and tuple-index-map for those rec-ids stored for table $t_j$ with $d_j <= d$ that is referenced by $t$

24: Perform 12 and 14

25: end if

26: end if

27: end for

28: Scan tables of $T$ for final mappings(rec-id$_1$,.....) to get the values of other attributes in the select statement of $Q$

29: return Result
Sample Query Processing

**Example 1**
```
select min(ps_supplycost)
from part, partsupp, supplier, nation, region
where r_name = 'EUROPE'
    and p_size = 15 and p_type like '%BRASS'
    and p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey;
```

\[ T = [\text{region}, \text{part}, \text{nation}, \text{supplier}, \text{partsupp}] \]
- Process region table to get the region key for ‘EUROPE’
- Process part table to get the part key for p_size = 15 and p_type like ‘%BRASS’
- Process regionkey-map to get the nation key tuple index
- Process nationkey-map to get the supplier key tuple index
- Process suppkey-map to get the partsupp key tuple index
- Process partkey-map to get the partsupp key tuple index
- Combine the results of above to get the final tuple index
- Process partsupplier table for final result
Map Size

PK-map and Tuple-index-map size:

- Size of 1st col ($S_1$) $\leftarrow$ (No. of rows of PK) * size of PK attribute value
- Size of other columns ($S_2$) $\leftarrow$ (No. of rows of PK) * size of record-id

Size of PK-map = $S_1 + \sum_{i=1}^{n} S_2[i] + c$

Size of Tuple-index-map = (No. of rows of FK) * (size of Integer)
Number of Communication

• No communication required when there is a join between two tables. We scan the maps instead.

• This reduces,
  – The number of communication during query execution
  – Total length of communication message for each query processing. Join operation access complete table or complete join attribute to perform join between two tables.
Experiment Setup (Small-Scale)

- Network of four virtual CentOS machines
  - 2.6GHz Six-Core AMD Opteron Processor
  - 2GB RAM
  - 10GB disk space
  - Oracle 11g

- Data generated using `dbgen` of TPC-H benchmark

- Total data size is 10GB

- Maps are horizontally partitioned and distributed among the nodes.

- Size of maps 10% – 12%
Test Queries (TPC-H)

Example 1: TPC-H Query 2 (page 30)

```sql
select s_acctbal, s_name, n_name, p_partkey,
       p_mfgr, s_address, s_phone, s_comment
from part, supplier, partsupp, nation, region
where p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'EUROPE'
  and ps_supplycost = ( select min(ps_supplycost)
                          from partsupp, supplier, nation, region
                          where p_partkey = ps_partkey
                          and s_suppkey = ps_suppkey
                          and s_nationkey = n_nationkey
                          and n_regionkey = r_regionkey
                          and r_name = 'EUROPE' )
order by s_acctbal desc, n_name, s_name, p_partkey;
```

Example 2: TPC-H Query 5 (page 37)

```sql
select n_name,
       sum(l_extendedprice*(1-1_discount)) as revenue
from customer, orders, lineitem,
     supplier, nation, region
where c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and l_suppkey = s_suppkey
  and c_nationkey = s_nationkey
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'ASIA'
  and o_orderdate >= date '1994-01-01'
  and o_orderdate < date '1994-01-01' +
    interval '1' year
  group by n_name
order by revenue desc;
```

Example 3: TPC-H Query 10 (page 46)

```sql
select c_custkey, c_name,
       sum(l_extendedprice*(1-1_discount)) as revenue,
       c_acctbal, n_name, c_address, c_phone, c_comment
from customer, orders, lineitem, nation
where c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate >= date '1993-10-01'
  and o_orderdate < date '1993-10-01' +
    interval '3' month
  and l_returnflag = 'R'
  and c_nationkey = n_nationkey
  group by c_custkey, c_name, c_acctbal, c_phone,
          n_name, c_address, c_comment
order by revenue desc;
```
Test Results

Comparison of average query execution time

Comparison of number of inter-node communications
Test Results

Comparison of total size of internode communications

Comparison of change in map size with the change in data size
Experiment Setup (Large-Scale)

• PlanetLab is a geographically distributed computing platform available as a testbed for deploying, evaluating, and accessing planetary-scale network services

  • Composed of 1050 servers at 400 sites (location) worldwide
  • We chose 50 machines worldwide running Red Hat 4.1 OS
  • Each machine has 2.33GHz Intel Core 2 Duo processor, 4GB RAM and 10GB disk space.
  • Installed MySQL on all of the machines
  • Generated 150GB of data using "dbgen" (by TPC-H) and distributed into all the machines
  • Generated proposed Maps, horizontally partitioned and distributed them into machines that contained corresponding data.
Test Queries

Query 1: Find the suppliers from ‘EUROPE’, who can supply given part type, and size at a minimum supply cost.

```sql
SELECT s_name, ps_supplycost
FROM region r, nation n, supplier s, part p, partsupp ps
WHERE r_r_name = 'EUROPE' AND p_p_type like '%BRASS' AND p_size = 15
    AND r_regionkey = n_regionkey AND n_nationkey = s_nationkey
    AND s_suppkey = ps_suppkey AND p_partkey = ps_partkey
HAVING min (ps_supplycost);
```

Query 2: Find all the suppliers from nation INDIA, who can supply part named ‘goldenrod’ of size 15.

```sql
SELECT s_name
FROM nation n, supplier s, part p, partsupp ps
WHERE n_n_name = 'INDIA' AND p_p_name like '%goldenrod%' AND p_size = 15 AND
    n_nationkey = s_nationkey AND s_suppkey = ps_suppkey AND p_partkey = ps_partkey;
```

Query 3: Find the total number of orders placed by the ‘UNITED STATES’ customers.

```sql
SELECT count (o_orderkey) as TotalOrders
FROM nation n, customer c, orders o
WHERE n_n_name = 'UNITED STATES' AND n_nationkey = c_nationkey
    AND c_custkey = o_custkey;
```

Query 4: Find the number of suppliers and customers in AFRICA.

```sql
SELECT count(s_suppkey) as TotalSuppliers, count(c_custkey) as TotalCustomers
FROM nation n, supplier s, customer c
WHERE n_n_name = 'AFRICA' AND n_nationkey = s_nationkey
    AND n_nationkey = c_nationkey
GROUP BY n_nationkey;
```

Query 5: Find the suppliers from JAPAN who can supply more than 9900 units of STEEL parts of brand #35.

```sql
SELECT s_name
FROM nation n, supplier s, part p, partsupp ps
WHERE n_n_name = 'JAPAN' AND p_p_type like '%STEEL' AND p_brand like '%#35'
    AND ps_availqty > 9900 AND n_nationkey = s_nationkey
    AND s_suppkey = ps_suppkey
    AND p_partkey = ps_partkey;
```
Test Results

• Comparison of time taken by queries 1-4
Test Results

- Comparison of time taken by query 5
- Comparison of num. of communications
Conclusion

• Proposed storage structures PK-map and Tuple-index-map to optimize Join operation in the Cloud environment
• Designed an algorithm for processing multi-join queries using the proposed storage structures.
• Experiment result shows the performance improvement during query execution
Future Work: Aggregate Optimization

• Goal:
  - Eliminate most of the sort and group-by operations with the help of integrity constraints and map structures (PK-map and Tuple-index-map)
  - Do not require pre-computation of aggregates as done in data cubes and materialized views.
References

• Swathi Kurunji, Ge Tingjian, Xinwen Fu, Benyuan Liu, Cindy Chen: "Optimizing Communication for Multi-Join Query Processing in Cloud Data Warehouses", International Journal for Grid and High Performance Computing (IJGHPC), Dec 2013

• Swathi Kurunji, Ge Tingjian, Benyuan Liu, Cindy Chen: "Communication Cost Optimization for Cloud Data Warehouse Queries", IEEE CloudCom, Dec 2012


Thank You

Questions?