CS 764: Topics in Database Management Systems
Lecture 5: Query Optimization-2

Xiangyao Yu
9/21/2020
Announcements

To ask a question in-class
- Raise your hand
- With the instructor’s permission, ask in the order of hand raising

Switch to Zoom?
- Lecture recordings will be uploaded to uwmadison.app.box.com
Discussion Highlights

EMP

<table>
<thead>
<tr>
<th>ENAME</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TCARD = 100
NCARD = TCARD * 100
DEPT.IDX_ENAME (clustered)
DEPT.IDX_DNAME (non-clustered)

# data pages
# tuples

SELECT ENAME
FROM EMP
WHERE DNAME = 'CS';

Q1: What are the possible access paths on EMP?
   1. Segment scan
   2. Clustered index scan on ENAME
   3. Non-clustered index scan on DNAME
Q2: Assume selectivity factor \( F = 1/10 \) for predicate DNAME='CS', which access path should be picked for the query above?

Segment scan cost = \( 100 / P \)

ENAME index scan cost = \( NINDEX(I) + 100 \)

DNAME index scan cost = \( (NINDEX(I)+10000) / 10 \)

(Assuming no caching in buffer pool)

Note: \( NINDEX(I) \neq 1\sim3 \)
An Overview of Query Optimization in Relational Systems

Surajit Chaudhuri
Microsoft Research
One Microsoft Way
Redmond, WA 98052
+1-(425)-703-1938
surajitc@microsoft.com

1. OBJECTIVE
There has been extensive work in query optimization since the early '70s. It is hard to capture the breadth and depth of this large body of work in a short article. Therefore, I have decided to focus primarily on the optimization of SQL queries in relational database systems and present my biased and incomplete view of this field. The goal of this article is not to be comprehensive, but rather to explain the foundations and present samplings of significant work in this area. I would like to apologize to the many contributors in this area whose work I have failed to explicitly acknowledge due to oversight or lack of space. I take the liberty of trading technical precision for ease of presentation.

2. INTRODUCTION
Relational query languages provide a high-level “declarative” interface to access data stored in relational databases. Over time, SQL [41] has emerged as the standard for relational query languages. Two key components of the query evaluation component of a SQL database system are the query optimizer and the query execution engine.

The query execution engine implements a set of physical operators. An operator takes as input one or more data streams and produces an output data stream. Examples of physical operators include:

- **Index Nested Loop**
  \[ \text{Index Nested Loop} \quad (A.x = C.x) \]

- **Merge-Join**
  \[ \text{Merge-Join} \quad (A.x = B.x) \]

- **Sort**

- **Table Scan**

The query optimizer is responsible for generating the input for the execution engine. It takes a parsed representation of a SQL query as input and is responsible for generating an efficient execution plan for the given SQL query from the space of possible execution plans. The task of an optimizer is nontrivial since for a given SQL query there can be a large number of possible execution plans.
Agenda

Query optimization components
  • Search Space
  • Cost estimation
  • Enumeration algorithm

Other considerations
Query optimization components

- SQL
- Parsing
- Query optimizer
- Execution engine
- Storage engine
- Relational Engine
 Query optimization components

**Search space** includes plans that have low cost

**Cost estimation** is accurate

**Enumeration algorithm** is efficient
Search Space

Search space of System R

• Linear sequence of joins
• Avoiding Cartesian products
• No discussion of outerjoins
• No discussion of group by
• No discussion of multi-block queries
Search Space – Join Order

left-deep tree

Convention: right child is the inner relation
Search Space – Join Order

Convention: right child is the inner relation
For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Search Space – Join Order

Convention: right child is the inner relation

For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Search Space – Join Order

Convention: right child is the inner relation
For nested-loop join or hash join, a left-deep tree allows tuples to be passed through pipelining
Bushy tree may produce cheaper plans but are rarely considered due to the explosion of search space
Search Space – Cartesian Product

System R defers Cartesian products after all the joins.
Evaluating Cartesian products early sometimes leads to cheaper plans.

- Example: dimension tables in OLAP in a *star schema*
System R defers Cartesian products after all the joins
Evaluating Cartesian products early sometimes leads to cheaper plans
• Example: dimension tables in OLAP in a star schema

* Figure from “An Overview of Data Warehousing and OLAP Technology”
One-sided outer joins are asymmetric and do not commute

- \( \text{Join}(R, S \text{ LOJ } T) = \text{Join}(R, S) \text{ LOJ } T \)

Repeatedly apply this rule to move outer joins after regular joins; regular joins can be freely reordered among themselves
Search Space – Group By

Example:

```sql
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples
Example:

\[
\text{SELECT D.name, count(*)}
\text{FROM EMP as E, DEPT as D}
\text{WHERE E.DeptID = D.DeptID}
\text{GROUP BY D.name}
\]

E has 10000 tuples
D has 100 tuples
Example:

```
SELECT D.name, count(*)
FROM EMP as E, DEPT as D
WHERE E.DeptID = D.DeptID
GROUP BY D.name
```

E has 10000 tuples
D has 100 tuples

Partial group by can also reduce cost
- Example: first aggregate total sales for all products, later aggregate sales for each division
- More on this topic in the group discussion
Search Space – Multi-Block Query

Merging nested subqueries

```
SELECT Emp.Name
FROM Emp
WHERE Emp.Dept# IN
    SELECT Dept.Dept# FROM Dept
    WHERE Dept.Loc='Denver'
    AND Emp.Emp# = Dept.Mgr

SELECT E.Name
FROM Emp E, Dept D
WHERE E.Dept# = D.Dept#
AND D.Loc = 'Denver' AND E.Emp# = D.Mgr
```

Nested query (correlated)  Unnested query
Merging nested subqueries

- Requires outerjoins when aggregation is present
- Nice body of work on doing this in an algebraic framework

```sql
SELECT Emp.Name
FROM Emp
WHERE Emp.Dept# IN
    SELECT Dept.Dept# FROM Dept
    WHERE Dept.Loc='Denver'
    AND Emp.Emp# = Dept.Mgr
```

```sql
SELECT E.Name
FROM Emp E, Dept D
WHERE E.Dept# = D.Dept#
AND D.Loc = 'Denver' AND E.Emp# = D.Mgr
```

Nested query (correlated)  Unnested query
Merging nested subqueries
  • Requires outerjoins when aggregation is present
  • Nice body of work on doing this in an algebraic framework

Semijoin-like techniques for multi-block queries
  • Send projected list from A to B to reduce the cost of evaluating B
  • More on semi-joins in distributed databases in later lectures

```sql
CREATE VIEW DepAvgSal As ( 
    SELECT E.did, Avg(E.Sal) AS avgsal 
    FROM Emp E 
    GROUP BY E.did
)

SELECT E.eid, E.sal 
FROM Emp E, Dept D, DepAvgSal V 
WHERE E.did = D.did AND E.did = V.did 
AND E.age < 30 AND D.budget > 100k 
AND E.sal > V.avgsal
```
Cost Estimation

System R: Cardinalities

Many commercial systems: histograms

- More buckets lead to higher accuracy but more memory/storage consumption
- Good only for single column: 2D histogram
Cost Estimation

System R: Cardinalities

Many commercial systems: histograms
  - More buckets lead to higher accuracy but more memory/storage consumption
  - Good only for single column: 2D histogram

Statistics collected through data sampling => error prone
  - Statistic errors propagate quickly. Can be disastrous

Cost computation
  - Many system parameters: hardware properties, data distribution, buffer utilization, data storage layout, etc.
Enumeration

Extensible optimizers (Example: Starburst and Volcano)
  • Add new join algorithms, new operators, new cost models

Volcano (powering SQL server)
  • Universal application of rules
    • Transformation rules: map one algebraic expression to another
    • Implementation rules: map algebraic expression to operator trees
    • Top-down dynamic programming technique
Other Considerations

Distributed databases
  • Communication cost

User defined function (UDF)
  • Hard to estimate the cost of a UDF

Materialized views
  • Reuse materialized views across queries
  • General problem undecidable

Miscellaneous
  • Mid-flight query re-optimization
  • Resources to consider (e.g., memory, power, cost)
  • Fuzzy queries in text/multimedia databases
Q/A – Query Optimization-2

Why unnest a query?
Modern query optimization?
  • Distributed/parallel, cloud, heterogeneous hardware
Why linear joins more common than bushy joins?
Given a query optimizer, does it matter how a query is written?
What is a star schema?
How does statistical information propagate?
Group Discussion

SELECT JOB.title, count(*) FROM JOB, EMP, DEPT WHERE JOB.jid = EMP.jid AND EMP.did = DEPT.did AND DEPT.location = "Madison" GROUP BY JOB.title

Consider only nested loop join and only the cost in terms of the # comparisons in the join (note that which relation is inner vs. outer in a join does not matter in this case)

Q1: If only one department is in Madison, what’s the cheapest plan?
   (hint: group-by can be partially pushed down)
Q2 [optional]: If all departments are in Madison, what’s the cheapest plan?

| EMP |   = 10000 tuples |
| DEPT | = 100 tuples |
| JOB |    = 10 tuples |
Before Next Lecture

Submit discussion summary to https://wisc-cs764-f20.hotcrp.com
- Title: Lecture 5 discussion. group ##
- Authors: Names of students who joined the discussion
- Summary submission Deadline: Tuesday 11:59pm

Before next lecture, submit review for