Research on the Query Performance Optimization Based on the DB2 UDB

Di LIU, Guihua LI

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Email: *liudi@uestc.edu.cn, lgh*6918@163.*com*

Abstract: Database tuning includes system optimization and query optimization, both are important to improve the performance of Database. This paper will present the query optimization based on DB2 UDB. There are multiple access paths for any given SQL statement because SQL is a declarative language. The database estimates the costs of each path and chooses what it thinks is the fastest. This paper will describe the basic methods of query tuning and how to tune a bad query to achieve the optimal performance.

Keywords: query optimization; access method; join method; filter factor; predicate

1 Introduction

There are many ways to Rome, but there are some ways that are faster, and some ways that are cheaper, or some ways that are both. Which one will you take?

Given a query, there are many plans that a database management system (DBMS) can follow to process it and produce its answer. All plans are equivalent in terms of their final output, but vary in their cost, i.e., the amount of time that they need to run. What is the plan that needs the least amount of time?

Such query optimization is absolutely necessary in a DBMS. The cost difference between two alternatives can be enormous. For example, consider the following database schema, which will be used throughout this chapter.

Execution Plans for a Given Query

```
SELECT name,floor
FROM   emp,dept
WHERE EMP.DNO = DEPT.DNOAND SAL > 100K
```

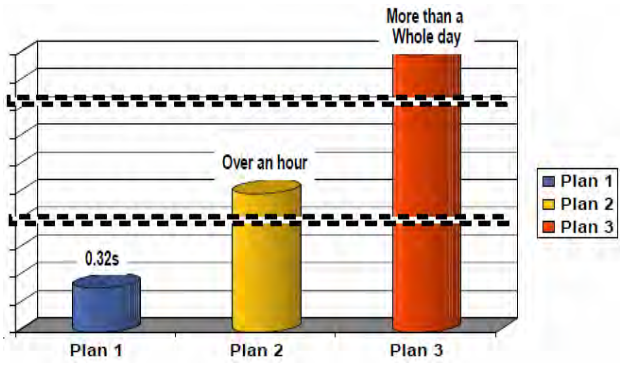

Figure 1. I/O costs comparison of 3 ways

Suppose we have a simple query on those tables, and DB2 has several options to execute it. Consider the following three different ways:

1) Through the B+-tree find all tuples of EMP that satisfy the selection on EMP.SAL. For each one, use the hashing index to find the corresponding DEPT tuples. (Nested loops, using the index on both relations.)

2) For each dept page, scan the entire EMP relation. If an EMP tuple agrees on the DNO attribute with a tuple on the dept page and satisfies the selection on EMP.SAL, then the EMP-DEPT tuple pair appears in the result. (Page-level nested loops, using no index.)

3) For each DEPT tuple, scan the entire EMP relation and store all EMP-DEPT tuple pairs. Then, scan this set of pairs and, for each one, check if it has the same values in the two DNO attributes and satisfies the selection on EMP.SAL. (Tuple-level formation of the cross product, with subsequent scan to test the join and the selection.)

Choosing the best access path for an SQL statement depends on a number of factors. These factors include the content of any tables that the SQL statement queries and the indexes on those tables.

In the Figure 1, calculating the expected I/O costs of these three ways shows the tremendous difference in efficiency. W1 needs 0.32 seconds, W2 needs a bit more than an hour, and W3 needs more than a whole day. Without query optimization, a system may choose plan W2 or W3 to execute this query with devastating results. Query optimizers examine all alternatives, so they should have no trouble choosing W1 to process the query.

Query optimization is of great importance for the performance of a relational database, especially for the execution of complex SQL statements. A query optimizer determines the best strategy for performing each query. Without a proper tuning of the queries, your database will never perform well, and lead to performance or availability problems, regardless of how fast your hardware is.

To a large extent, the success of a DBMS lies in the quality, functionality and sophistication of its query optimizer, since it determines much of the system's performance.

978-1-935068-12-9

2 Query Optimization in Db2

2.1 Access Paths

Access to DB2 data is achieved by telling DB2 what to retrieve, not how to retrieve it. Regardless of how the data is physically stored and manipulated, DB2 and SQL can still access that data. This separation of access criteria from physical storage characteristics is called physical data independence.

Access paths are a significant factor of DB2 performance. An access path is the path that DB2 uses to locate data that is specified in SQL statements. The access path that DB2 chooses determines how long the SQL statement takes to run.

In any given SELECT statement, there are often many different access paths to consider. The following Figure 2 shows some factors that can lead to multiple access paths:

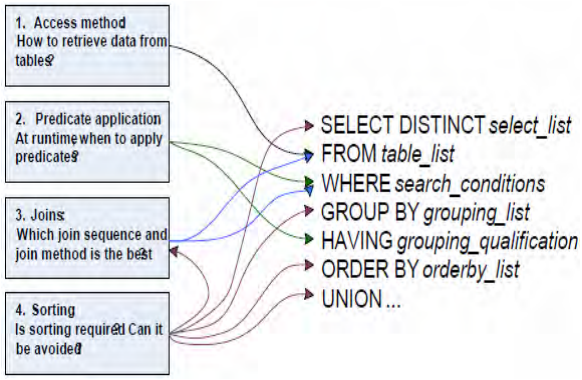

Figure 2. Access Path Strategies

Choosing the best access path for an SQL statement depends on a number of factors. Those factors include the content of any tables that the SQL statement queries and the indexes on those tables. DB2 also uses extensive statistical information about the database and resource use to make the best access choices.

2.2 Table Access Methods

Three table access methods, which are table space scan, index access and list prefetch.

Generally, the fastest way to access DB2 data is with an index. Indexes are structured in such a way as to increase the efficiency of finding a particular piece of data. However, the manner in which DB2 uses an index varies from statement to statement. DB2 uses many different internal algorithms to traverse an index structure. These algorithms are designed to elicit optimum performance in a wide variety of data access scenarios.

When an index is not used to satisfy a query, the resulting access path is called a table space scan, as opposed to an Index Scan. A table space scan performs page-by-page processing, reading every page of a table space (or table).

Sequential prefetch is a read-ahead mechanism invoked to prefill DB2's buffers so that data is already in memory before it is requested. The optimizer requests sequential prefetch when it determines that sequential processing is required. The sequential page processing of a table space scan is a good example of a process that can benefit from sequential prefetch.

2.3 Join Methods

When more than one DB2 table is referenced in the FROM clause of a single SQL SELECT statement, a request is being made to join tables. The optimizer has a series of methods to enable DB2 to join tables.

Multi-table queries are broken down into several access paths. The DB2 optimizer selects two of the tables and creates an optimized access path for accomplishing that join. When that join is satisfied, the results are joined to another table. This process continues until all specified tables have been joined.

When joining tables, the access path defines how each single table will be accessed and also how it will be joined with the next table. Thus, each access path chooses not only an access path strategy (for example, a tables pace scan versus indexed access) but also a join algorithm. The join algorithm, or join method, defines the basic procedure for combining the tables. DB2 has three basic methods for joining tables:

- Nested loop join
- Sort merge join
- Hybrid join

The most common type of join method is the nested loop join. A qualifying row is identified in the outer table, and then the inner table is scanned searching for a match. When the inner table scan is complete, another qualifying row in the outer table is identified. The inner table is scanned for a match again, and so on.

In the sort merge join, the tables to be joined are ordered by the keys. This ordering can be the result of either a sort or indexed access. After ensuring that both the outer and inner tables are properly sequenced, each table is read sequentially, and the join columns are matched. Neither table is read more than once during a merge scan join.

Hybrid join can be a good join choice when the inner table has an index defined on the join column, and there are duplicates on the outer table on the join column.

NOTE: Hybrid joins apply to inner joins only. Either of the other two join methods can be used for both inner and outer joins.

2.4 Predicate Application

Predicate is found in the WHERE, HAVING or ON

clauses of SQL statements, it describe the attributes of the data. Only WHERE and ON predicates impact access path selection. Figure 3 shows the predicate architecture.

Classification of predicate "type" includes categories such as =, <, >, <=, >=, Null, Between, Like, In List, Not and Arithmetic predicates, as well as Subquery form In, Any, All, Some, Exists, in both the correlated and non-correlated varieties.

Predicate "attributes" dictate when they are processed (either at Stage1 (DM) or Stage2 (RDS) time), whether they are indexable or nonindexable (the ability to match against values in the index key), if they are Boolean Terms (which can qualify or reject rows standalone), and whether they are local predicates or join predicates.

Indexable predicates can match index entries and not indexable predicates cannot match index entries. Stage1 predicates, or Sargable, can be processed by the Data Manager (DM), the 1st stage of predicate processing. All indexable predicates are also stage1, but not all stage1 predicates are indexable. Stage2 predicates, or Residudal, can be done by RDS (Relational Data System). The cost of processing for stage2 is greater than stage1.

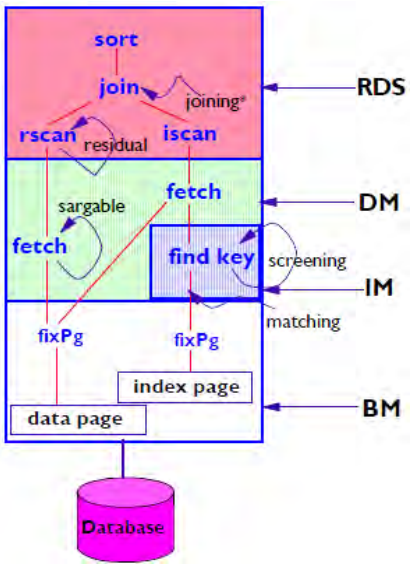

Figure 3. Predicate Architecture

2.5 Filter Factor

Filter factor is a ratio (a number between 0 and 1) that is used to estimate CPU and I/O costs. It is an estimate of the proportion of table rows for which a predicate is true. If you remember nothing else about filter factors, remember this: The lower the filter factor, the lower the cost and, in general, the more efficient your query will be.

Three factors that affect Filter Factor Estimation:

a) Cardinality

It is the number of distinct values for a column. We can use RUNSTATS command to collect column cardinality in a table:

RUNSTATS TABLESPACE (DBNAME.TSNAME)
TABLE (ALL or PAT_TABLE)
COLUMN(ALL or <list of columns>)

b) HIGH2KEY/LOW2KEY

They are the second highest/lowest values in this column. They are Interpolations used to estimate range predicates, like 'between, <, <=, >, >='.

c) Frequency

The frequency describes the frequency of a certain value in a column. It describes the skew of data by providing non-uniform data distribution information.

3 The Implementation of Query Optimization

Until now, we have seen that for any given SQL statement, there are many access paths to consider. The database estimates the costs of each path and chooses what it thinks is the fastest.

The two main steps in query optimization:

-Enumerate the possible evaluation plans for the query.

-Estimate the cost of each plan, choose the fastest.

We need to keep a balance between bind time performance (Time to figure out the access path) and run-time performance (Time to execute the access path).

When improving predicate application, we often try promoting predicates to an earlier stage for performance. Indexable is more efficient than sargable, and sargable is more efficient than residual. In the following, table 1 shows some examples:

Table 1. promoting predicates to an earlier stage

Residual	Sargable	Indexable
DEC_COL = :inthost		DEC_COL = DECIMAL(:inthost)
QTY * 2 = :dechost		QTY = :dechost/2
YEAR(DATECOL) = 2003		DATECOL BETWEEN '2003-01-01' AND '2003-12-31'
:host BETWEEN C1 AND C2		:host >= C1 AND :host<=C2
DATECOL + 10 YEARS < CURRENT DATE		DATECOL < CURRENT DATE-10 YEARS
LOCATE('P',FIRSTNAME) >0	FIRSTNAME LIKE '%P%'	
	DATECOL <> '9999-21-31'	DATECOL < '9999-21-31'
	GENDER <> 'F'	GENDER = 'M'

978-1-935068-12-9 © 2010 SciRes.

4 Other Influence Factors

4.1 Parallelism

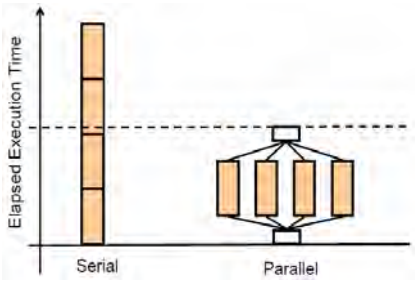

Figure 4. Serial VS Parallel

Another technique that can be applied by the optimizer is query parallelism. When query parallelism is invoked, DB2 activates multiple parallel tasks to access the data. A separate subtask MVS SRB is initiated for each parallel task. Both partitioned and non-partitioned table spaces can take advantage of query parallelism. From Figure 4, we can see it clearly, query serial spend mort time than query parallelism in the same condiction.

Based on the different processing power to be executed concurrently, there are three types of query parallelism that DB2 can perform: Query I/O parallelism, Query CP parallelism, Query Sysplex parallelism.

4.2 Optimize for N Rows

The OPTIMIZE FOR n ROWS clause minimizes overhead for retrieving few rows and lets an application declare its intent to do either of these things:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR n ROWS clause to choose access paths that minimize the response time for retrieving the first few rows. For distributed queries, the value of n determines the number of rows that DB2 sends to the client on each DRDA network transmission.

OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. This means that whenever possible, DB2 avoids any access path that involves a sort. If you specify a value for n that is anything but 1, DB2 chooses an access path based on cost, and you won't necessarily avoid sorts.

5 Conclusions

In this paper, we have discussed the importance of DB2 query optimizer, the many methods for query optimization and implementation. We all know DB2 is an important database product of IBM, and IBM has invested heavily in Optimizer technology, such as Optimization Service Center and OMEGAMON DB2 Performance and so on, which are very powered performance analysis tools that IBM developed to improve SQL performance. So we've barely scratched the surface here, the overall mechanism is extremely complex for query optimization.

References

[1] Frank J. Ingrassia, "Emerging DB2 Optimizer Technology", Computer Measurement Group's 1995 International Conference.
[2] IBM Corp, "DB2 Version 9.1 for zOS Performance Monitoring and Tuning Guide", SC18-9851-04.
[3] N. Kabra and D. DeWitt, "Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plan", SIGMOD, 1998.
[4] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, "Access Path Selection in a Relational Database Mana- gement System", SIGMOD 1979, pp. 23-34.
[5] N. Swami, K. B. Schiefer, "On the Estimation of Join Result Sizes", EDBT 1994, pp. 287-300.
[6] IBM Crop, "Application Programming and SQL Guide_V9 for zOS", SES1-2937-04.
[7] P. Gassner, G. M. Lohman, and Y Schiefer, B. Wang, "Query Optimization in the IBM DB2 Family", Data Engineering Bulletin, 16(4), 1993.