Plan-Per-Tuple Optimization Solution – Parallel Execution of Expensive User-Defined Functions

Felipe Cariño and William O'Connell

NCR Teradata – Parallel Systems
100 N. Sepulveda Blvd. El Segundo, CA 90245
{fc1,wto}@ElSegundoCA.NCR.com

Abstract

Object-Relational database systems allow users to define new user-defined types and functions. This presents new optimizer and run-time challenges to the database system on shared-nothing architectures. In this paper, we describe a new strategy we are exploring for the NCR Teradata Multimedia Database System; our focus is directing research for real applications we are seeing. In doing so, we will briefly describe optimizer challenges particularly related to predicate use of large multimedia objects, such as video/audio clips, images, and text documents. The motivation for this work is based on database tuning [SD96] for diverse queries related to multimedia objects. Most notably, expensive and/or high variant user defined functions [Hel98].

Our approach is referred to as *plan-per-tuple*. The primary focus being on large objects used as predicate-based terms when a non co-located join is involved in the query. But can also be applicable in non co-located join scenarios also. The execution engine can choose from among N! resource optimization strategies; where N represents system manageable resources. In our case, the N resources are: (i) interconnect saturation levels, (ii) available physical memory, (iii) CPU utilization, and (iv) available disk spool space percentages. However, this technique can be applied to any system resources being managed. The optimizer search space does not include these N! resource optimization strategies per'se, these are execution engine run-time optimization strategies. When the optimizer identifies expensive, or more importantly a high variant, user-defined function in the predicate (via collected statistics), then the optimizer can generate plans that incorporate *plan-per-tuple* optimization for that particular compiled query. When executing the plan, a different execution strategy can be used per tuple; the available execution choices do not necessarily equal *N!* We describe when such an overhead for run-time selection is acceptable.

1.0 Introduction

With the enhancement of SQL3's typing system, users are allowed to define new user-defined types and functions. In fact, many of our initial customer installations are employing large objects (out-of-line tuple attributes) into their applications; this *includes* large objects used as predicate-based terms [CS98]. This presents new optimizer and run-time challenges to the system on a shared-nothing MPP-based architectures. In this paper, we outline the strategy that we have been exploring in the NCR Teradata Multimedia Database System (M-DBS) in which development started in 1993 [CS98, Car98, CSI94, OIS+96]. The M-DBS Release 1 was made generally available in 1997. This optimization approach was not incorporated into this release. The system's fundamental design is based on NCR Teradata Database System (DBS) version 2 (V2). For in-depth discussions on the DBS, see [CK92, CSK95, WCP93].

M-DBS extends the DBS with SQL3 capabilities [Sto96]. A comprehensive analysis of the M-DBS architecture, key new concepts, and user-defined type (UDT) and user-defined function (UDF) optimizer issues can be found in [CSI94, CS98]. Many previous papers analyzed diverse optimization strategies, which include: cost- based [SAC+79], rule-based [Fre87], extensible [PHH92], libraries [MBH+96], multimedia [SG96], expensive UDFs [Hel98], and UDF predicate re-writes [CS96].

 Proceedings of the 24th VLDB Conference New York, USA, 1998

This paper focuses on overviewing dynamic execution strategies [CG94] based on optimizer statistical decisions; most notably, predicate terms which are most likely defined as large object (LOBs) columns. A LOB is any column that does not fit in-line in the tuple. We will additionally highlight some of the complex UDF issues that arise in shared-nothing architectures; in particular, for non co-located joins when LOBs are in the predicate; this implies LOBs may be moved.

As previously noted, we are concentrating on a run-time execution strategy based on optimizer statistics, not optimizer search strategies used when compiling the query. For a comprehensive discussion on optimizer search strategies with expensive predicate terms, see [Hel98]. The optimizer does not know, nor can it, the complete state of the system when the plan is executed while compiling the query. Moreover, compiled plans are many times cached for latter execution.

Additionally, Teradata DBS installations commonly have hundreds to thousands of sessions currently connected to the DBS running typically tens of queries at anyone instant. Flow and load control management as well as systems resource management is critical; system resource must be utilized at 100%, but no more.

Modifying the behavior of a compiled plan based on the system state when it is executed can assist the execution engine in (i) keeping all critical resources at or near peak capacity, and (ii) controlling thrashing through data flow and resource management. As a result, our plan-per-tuple solution addresses the presence of variance, hot spots and diverse workloads on the system. In this case, variance is measured in main memory, interconnect saturation, CPU, and disk spool space utilization.

The paper is organized as follows:

Section 2 briefly overviews the DBS V2 optimizer. Section 3 overviews the M-DBS optimization issues. Section 4 contains a brief description of our plan-per-tuple run-time optimization strategy. Finally, concluding remarks are in section 5.

2.0 Teradata Database V2 Optimizer Overview

We will first briefly overview a few of the attributes of the DBS V2 optimizer. A detailed discussion is beyond the scope and purpose of this paper. The DBS has a mature cost-based optimizer for handling SQL-92 queries; rules and re-writes are built-in and cost based. It handles simple stored procedures (persistent stored modules), but does not handle UDFs. Its optimization strategy is based on system throughput. In doing so, it considers three system resources while evaluating *all* execution strategies at compile time[1], which are CPU, disk usage, interconnection network usages, respectively. It relies on *"selectivity"* estimates when invoking its internal formulas; statistics are the key.

There are several other important factors such as indexing, table demographics and other statistical information that is kept or generated at query compile-time. Memory is not directly part of the optimizer formulas (it's correlated to other resources); at compile time it is difficult to predict what memory usage will be when a query is executed and what the memory contention will be during an execution. Our model attempts to address this by using memory utilization considerations at run-time, not only at compile or scheduling time.

If the table has no statistics, the optimizer will generate table demographics by random sampling of the table data blocks. If the user has collected statistics on the involved tables, the join plan will always be consistent. The cost model used minimizes the total resource utilization, assuming exclusive use of the system. The optimizer minimizes the sum of the CPU utilization, the disk array utilization (calculated using maximum disk array throughput), and BYNET interconnect utilization.

The optimizer also determines whether the use of an index is more efficient than a full file scan over the data. But, all this is compile time considerations, it can not consider what other queries may be running on the system when this query is executed. We believe that under certain UDF invocation classifications (via statistics), run-time considerations should be taken. Especially when LOBs are used as a joining or filtering term in a non co-located join operation.

[1] Most poor strategies are pruned very quickly.

Moreover, our focus is on maintaining peak performance on all managed system resources, not a subset. The ultimate goal is throughput. We will discuss this further in the subsequent sections.

3.0 High Variance/Expensive UDF Run-Time Optimization Issues

As previously stated, our analysis is for shared-nothing architectures focusing on high variance or expensive UDFs[2]; this typically implies LOBs are involved. We address several solutions for handling (i) data skew and load balancing when LOB columns are within the predicate, and (ii) maximizing total system throughput optimization under heavy loads.

In general, SQL optimizers require and/or use cost estimation, tuple statistics, indexes and/or sampling to optimize queries. Large multimedia databases where LOBs are utilized in predicates (via UDFs) introduce new load balancing and skew problems for *any* one of the managed resources.

Cost estimation of UDFs work when the average execution time has low variance, but can be problematic when there is high variance. Consider a UDF that does content analysis on a video column where video lengths in the column span from 2 minutes in length to over 2 hours. Note that this says nothing about the skew pattern within the column itself relative to distribution of sizes of videos. Sampling implies running a random instance of every UDF in the query and using this information to generate an efficient query plan.

Sampling is easy to implement, but assumes that a UDF execution time is uniform across all objects. If sampling is used on a UDF with large execution cost variance, then non-optimal plans may be generated. Allowing the execution engine to adapt to the characteristics of each tuple can help reduce load balancing and skew problems.

We must realize that regardless of the techniques used to generate a query plan (e.g. sampling or historical data), if the cost variance varies widely per tuple, then an efficient (optimal) plan may not be possible to generate at compile-time. Also, an efficient execution plan depends on how the system resources (CPU, memory, disk and network) are being used by all active queries. This affects not only high variant UDFs, but expensive uniform ones too. This variance (or expense) is relative to any one or more managed system resources.

Lets consider the video column example above. Also, lets assume that there were other predicate terms as well as a join being involved causing the video column UDF to be potentially executed on a non co-located node (with respect to the base table LOB's physical location). If the interconnect has low utilization at execution time, then pushing all LOBs with the rows may be appropriate assuming high probability that the UDF will actually be invoked on all rows (the optimizer has statistics of this probability); else pulling may be more appropriate.

However, if interconnect utilization is high, then the small LOBs (e.g., a few minutes in length) can be pushed while the large ones pulled. Alternately, the tuple's column evaluation can temporally migrate to the physical location of the LOB. Then, the result is sent back. Note, no LOB movement was done in this case. However, the choice of these latter two again is based on the system's state. Actually in this join example, there are really several strategies that can be used. They are summarized here:

- Push all LOBs from the source to sink along with redistributing the actual rows.
- Pull all LOBs (portion or whole) from source to sink; pulled portions can be written to local disk after pulling or discarded based on expected access patterns and/or available spool space.
- Intermix the pushing and pulling approaches based on the LOB size while re-distributing.
- Migrate tuple computation to location of LOB, process UDF there, then ship result back. This is function-shipping which prevents LOB movement over interconnect. Moreover, if two LOBs are required by a UDF (such as a joining predicate term), this may entail migration to one LOB site and pulling the other LOB.
- Re-decluster the joined result-set based on the partitioning strategy of the LOB. This also eliminates LOB movement; the UDF is invoked after the redistribution.

[2] Variance and expense with respect to an attribute's memory, interconnect, CPU, and disk spool space usage.

Moreover, many join algorithms look at some tuples more than once in the joining phase. In cases where these tuples include LOBs being pulled, it is optimal to only pull them once. But, if disk spool space is low and there exists suitable interconnect bandwidth to continually stage the LOBs, then pulling for each reference may be appropriate. The same is true on main memory caching of LOBs. The default is no caching due to sequential flooding. However, if memory utilization is low and the LOB data is not excessive, then caching of LOB data may be appropriate[3], this would eliminate allocating spool space in cases were the LOB was expected to be accessed multiple times during the joining phase. Table statistics are needed along with the current system's state to make these decisions.

The goal of the M-DBS optimizer is to balance the diverse resources and enhance either query response-time or throughput. The optimizer can either generate a query execution plan at compile-time (low variance, uniform system use) or generate a plan that indicates it can use dynamic execution based on resources. Both options utilize the same optimizer search strategies at compile time when generating the execution plan, but the latter allows the execution engine to make run-time decisions.

4.0 Plan-Per-Tuple Run-Time Optimization Strategy

The key items deal with: resources, selectivity and statistical variance. In shared-nothing architectures, the four key system resource factors: (i) cpu, (ii) memory, (iii) disk and (iv) network that must be managed for either response time or throughput.

We will denote resource usage for each of these resources by: Rcpu, Rmem, Rdisk and Rnet, respectively.

The optimizer can have statistical knowledge of memory resource usage of a UDF on a LOB column, for example a UDF typically may access heap and/or buffer spool space while processing a LOB, these resources are tracked on the DBS. Moreover, CPU and disk accesses of a LOB[4], as well as disk spool space required per invocation are also tracked. This indirectly indicates interconnect cost if the LOB must be used.

[3] The alternative is to abort the transaction due to lack of disk resources spool space.

[4] Percentage of actual object accessed by UDF, such as 10%.

In processing the plan-per-tuple strategy, the optimizer can either generate a compile-time only plan (as done today), or place directives directly in the compiled plan to provide run-time options. The latter is only done when high-variant and/or expensive UDFs are involved in the predicate evaluation. In this case, it makes sense to generate "plan-per-tuple" series of execution plans.

The idea behind a "plan-per-tuple" (Figure 1) is that the M-DBS Evaluator (which executes the query plan) checks that state of the (SMP) node for resource availability and then selects a plan with that resource as the dominant factor. For example, a query plan, that optimizes resources in cpu, disk, memory and network usage denoted by Plan (Rcpu, Rdisk, Rmem, Rnet) would be selected based on the resource availability for the (SMP) node.

The resource state per node is kept and maintained by a Global Resource Object that is queried at execution by the evaluator in order to select an execution plan.

As a result, if the variance is low, then a compile-time only plan is generated. Otherwise, for non-uniform UDF execution - which may have variance in UDT object values - numerous directives are added on a per access module processor (AMP) Step basis.

The query executor considers these directives when processing the AMP Step. Four logical things are required when processing the step, they are (i) UDF statistical information applicable to the column, (ii) the tuple attribute information (such as size and location), (iii) applicable system resource utilization levels (of some reasonable granularity), and (iv) the plan-per-tuple directives.

To evaluate the plan-per-tuple optimization, the order of importance of each resource must be determined (e.g. Rmem, Rcpu, Rnet, Rdisk); this may be different for any two tuples in the operation. In this case, there are 4! (24) ways to organize resource priorities. The goal must be to quickly select a plan that minimizes total resource usage for that tuple, yet considers existing utilizations.

This execution model is only feasible when the predicate evaluation contains expensive UDFs. It works best when there is a high variance in some resource usage among the tuple attributes that make up the table. The variance is what really allows the alternative choices to be exploited.

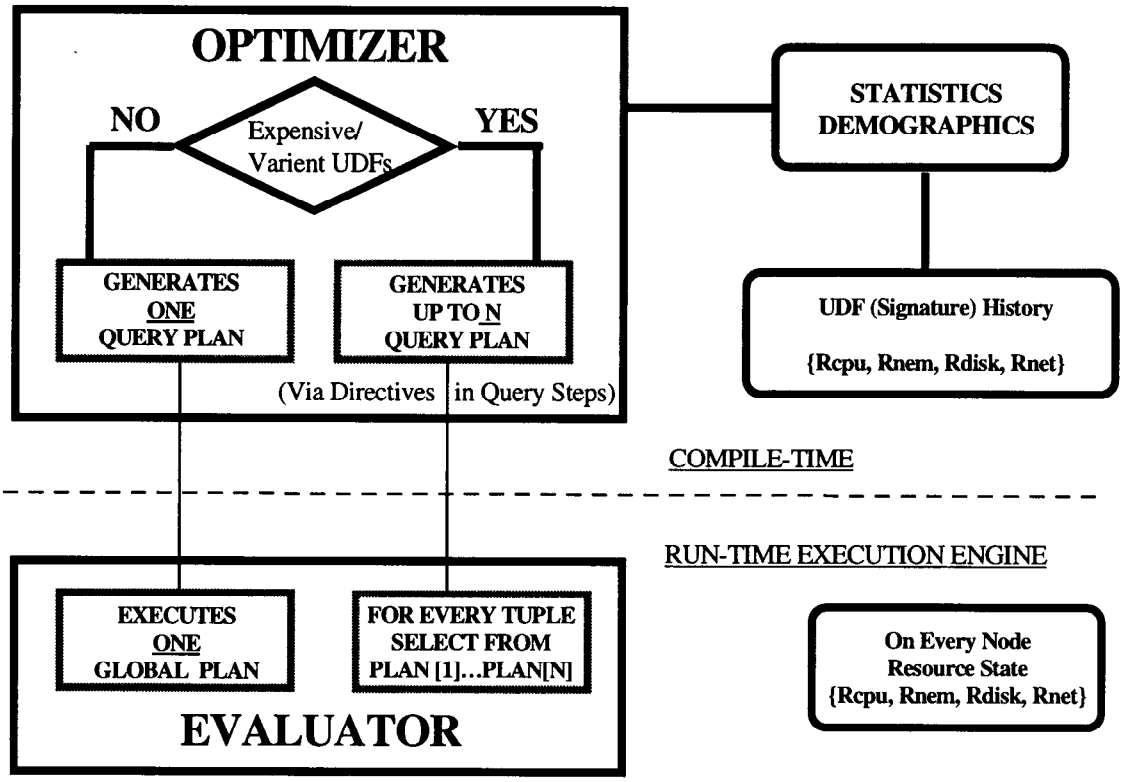

Figure 1: Plan-Per-Tuple Execution Flow

5.0 Conclusion

This paper described the major UDF issues that Object-Relational database optimizers and DBA must handle when optimizing expensive high variant UDFs. This variance does not only involve the algorithm used in the UDF, but also the data values stored in a table's column (e.g., complexity, size, or density).

The Teradata philosophy is that the database must handle optimizations with minimal DBA (e.g. no pragma) hints. The diverse and complex SQL3 applications that customers are now approaching us on have led us to develop a plan-per-tuple approach to address highly diverse OR/DBMS UDF queries that high high/low skew and variance. We are continuing to explore its merits.

6.0 Acknowledgments

The authors would like to thank the anonymous referees, Bill McKenna, Pekka Kostamaa, Warren Sterling and Dave Schrader for their suggestions, comments and help.

7.0 References

[Car98] Cariño, F., "Teradata Goes Full Color", Teradata Review, January 1998

[CG94] Cole, R. L. and Graefe, G., "Optimization of Dynamic Query Evaluation Plans", Proceedings of the ACM SIGMOD, pp. 150-160, June 1994.

[CK92] Cariño, F. and Kostamaa, P., "Exegesis of DBC/1012 and P-90", Proceedings of the 4th International Parallel Architectures and Languages Europe (PARLE `92), Springer-Verlag, pp. 877-892

[CSI94] Cariño, F., Sterling, W. and Ieong, I.T., "Teradata-MM - A Complete Multimedia Database Solution", ACM Multimedia Workshop on Multimedia Databases, San Francisco, California, October 21, 1994.

[CSK95] Cariño, F., Sterling, W. and Kostamaa, P., "Industrial Database Supercomputer Exegesis - The DBC/ 1012, The NCR 3700, The Ynet and The BYNET", Emerging Trends in Knowledge and Database Systems, IEEE Computer Society Press, Los Alamitos, California, pp. 139 - 157, 1995.

[CS98] Cariño, F. and Sterling, W., "Parallel Strategies & New Concepts for a Petabyte Multimedia Database Computer", Parallel Database techniques, IEEE Computer Society Press, Los Alamitos, California, 1998.

[CS96] Chaudhuri,. S. and Shim. K., "Optimization of Queries with User-Defined Predicates", Proceedings of the 19th International Conference on Very Large Databases (VLDB `96), pp. 87 - 98.

[Fre87] Freytag, J.C., "A Rule-Based view of query optimization", Proceedings of the ACM SIGMOD, pp. 173-180, June 1987.

[HS93] Hellerstien, J.M and Stonebraker, M., "Predicate Migration: Optimizing Queries with Expensive Predicates", Proceedings of the ACM SIGMOD, pp. 267 - 276.

[Hel98] Hellerstein, J.M., "Optimization Techniques For Queries with Expensive Methods", To appear, ACM Transactions on Database Systems (TODS).

[MBH+96] McKenna, W., Burger, L. Hoang, C. and Troung, M. "EROC: A toolkit for Building NEATO Query Optimizers", Proceedings of the 19th International Conference on Very Large Databases (VLDB `96), pp. 99 - 110, 1996.

[OIS+96] O'Connell, W., Ieong, I.T., Schrader, D. and et al., "Prospector: A Content-Based Multimedia Object Server for Massively Parallel Architectures", Proceedings of the ACM SIGMOD, pp. 68 - 78, 1996.

[PHH92] Pirahesh, H, Hellerstein, J. M. and Hasan, W., "Extensible/rule-based query rewrite optimization in Starburst", Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD 92), pp. 39-48, June 1992.

[SAC+79]Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A. and Price, T.G., "Access path selection in a relational database management system", Proceedings of the ACM SIGMOD, pp. 23 - 34, 1979.

[SG96] Surajit, C. and Gravano, L. "Optimizing Queries over Multimedia Repositories", Proceedings of the ACM SIGMOD, pp. 91 - 102, 1996.

[SD96] Shasha, D. "Database Tuning Book: A Principled Approach", Prentice Hall, Englewood Cliffs, New Jersey, ISBN 0-13-205246-6, 1992.

[Sto96] Stonebraker, M., "Object/Relational DBMSs: The Next Great Wave", Morgan Kaufmann San Francisco, California, ISBN 1-55860-397-2, 1996.

[WCP93] Witkowski, A., Cariño, F. and Kostamaa, P. "NCR 3700 - The Next-Generation Industrial Database Computer", Proceedings of the 19th International Conference on Very Large Databases (VLDB `93), pp. 230 - 243.