2012 IEEE 28th International Conference on Data Engineering

Analyzing Query Optimization Process: Portraits of Join Enumeration Algorithms

Anisoara Nica [#1], Ian Charlesworth [*], Maysum Panju [*]

[#]*Sybase, An SAP Company*
Waterloo, Ontario, Canada
[1]anica@sybase.com

[*]*David R. Cheriton School of Computer Science, University of Waterloo*
Waterloo, Ontario, Canada

***Abstract*—Search spaces generated by query optimizers during the optimization process encapsulate characteristics of the join enumeration algorithms, the cost models, as well as critical decisions made for pruning and choosing the best plan. We demonstrate the *JoinEnumerationViewer* which is a tool designed for visualizing, mining, and comparing plan search spaces generated by different join enumeration algorithms when optimizing same SQL statement. We have enhanced Sybase SQL Anywhere relational database management system to log, in a very compact format, its search space during an optimization process. Such optimization log can then be analyzed by the *JoinEnumerationViewer* which internally builds the logical and physical plan graphs representing complete and partial plans considered during the optimization process. The optimization logs also contain statistics of the resource consumption during the query optimization such as optimization time breakdown, for example, for join enumeration versus costing physical plans, and memory allocation for different optimization structures. The SQL Anywhere Optimizer implements a highly adaptable, self-managing, optimization process by having several join enumeration algorithms to choose from, each enhanced with different ordering and pruning techniques. The emphasis of the demonstration will be on comparing and contrasting these join enumeration algorithms. The demonstration scenarios will include optimizing SQL statements under various conditions which will exercise different algorithms, pruning and ordering techniques. The generated search spaces and statistics will then be visualized and compared using the *JoinEnumerationViewer*.**

I. Introduction and Motivation

SQL Anywhere[1][1] is an ANSI SQL-compliant RDBMS designed to run on a variety of platforms from server-class installations to mobile devices using the Windows Mobile operating system. SQL Anywhere is a self-managing RDBMS with high reliability, high performance, synchronization capabilities, small footprint, and a full range of SQL features across a variety of 32- and 64-bit platforms.

The SQL Anywhere Optimizer is designed to be highly adaptable to any type of query workload, system resources available to the optimization process, and the state of the database server. The SQL Anywhere Optimizer [2] dynamically chooses, based on the query graph characteristics and on the resources available to the optimization process, one of the join enumeration algorithms implemented in SQL Anywhere. The cost-based join enumeration algorithms available for the optimization process include dynamic programming algorithms over the bushy trees as well as depth-first search enumeration algorithms over left-deep trees. A query is represented internally as a $\mathcal{PSNS}$-annotated[2] normalized join operator tree [3] where each normalized join corresponds to a table expression in a null-supplying side of an outerjoin, or to a table expression of a query block. The optimization process is invoked for each normalized join, in a bottom-up fashion. Each invocation of the optimization process starts by obtaining an initial estimated cost using the cheap *backtracking* algorithm. Based on this initial cost, the system resources, and the properties of the query graph, an enumeration algorithm is chosen for that optimization process. All algorithms work with the same hypergraph representation of a normalized join and share data structures and classes used during enumeration such as the memoization table, special memory heaps, the cost model, etc. (The join enumeration algorithms supported in SQL Anywhere are described in Section III.) Given these unique characteristics of the SQL Anywhere Optimizer, the optimization logs generated during query optimization by different join enumeration algorithms are very suitable for contrasting the join enumeration algorithms from point of view of resource consumption, enumerated logical plans, costed physical plans, pruning and ordering techniques.

A critical aspect of the join enumeration algorithm is the order in which the access plans are generated and the pruning strategies implemented for a particular join enumeration algorithm. In general, all algorithms generate a partial physical access plan, incrementally compute its estimated cost, and prune it when the cost exceeds the cost of the best plan found so far for the same table expression, or for the best complete plan. It is thus critical to generate the best plans very early in the search such that pruning will be done aggressively later on. Furthermore, some of the enumeration algorithms have specific heuristics to rank and prune partial access plans based on their logical properties. For example,

[1]Sybase and SQL Anywhere are trademarks of Sybase Inc. Other company or product names referenced in this paper are trademarks and/or servicemarks of their respective companies.

[2]$\mathcal{PSNS}$ = Preserved Side / Null-supplied Side

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.132

the proprietary *backtracking* algorithm uses an enumeration governor which heuristically, based on the enumeration order, prunes unpromising subspaces. Hence, the performance of the optimization process as well as the quality and robustness [4] of the best final plan depend on the heuristics used to decide the order in which the access plans are enumerated. The *JoinEnumerationViewer* system was designed specifically to visualize and analyze the characteristics of a join enumeration algorithm which affect both the quality of the best plan and the resource consumption (e.g., elapsed time, buffer pool utilization) during the optimization process.

One motivation for the design of the *JoinEnumerationViewer* system is a recent series of seminal papers [5], [6], [7], [8] proposing new join enumeration algorithms with the main goal of efficiently and exhaustively enumerating valid logical partitions over bushy-trees without cross-products. The purpose of these algorithms is to optimize the time spent for the logical partition enumeration. However, in our experience, and also according to the work presented in [9], the logical partition enumeration is just a small fraction of the total optimization time. The work presented in [9] puts forward interesting assumptions about the optimization time breakdown in the DB2 optimizer. As the SQL Anywhere Optimizer was enhanced with new join enumeration algorithms including three algorithms based on the work described in [6], [7], and [8], the *JoinEnumerationViewer* system was architected to compare the characteristics of these algorithms. By recording the resource consumption statistics in the optimization logs, we can analyze the statistics on optimization time breakdown and memory required for each of the join enumeration algorithms implemented in the SQL Anywhere Optimizer. Another source of inspiration for this work is the *SearchSpaceAnalyzer* system which was demonstrated at ACM SIGMOD 2009 [10]. *SearchSpaceAnalyzer* supports the visualization of search spaces generated by a backtracking algorithm over the left-deep trees only, where the search spaces must be ordered forests of rooted ordered trees.

A visualizing tool which deals only with complete best plans of a parameterized query is the Picasso system [11]. The system can visualize a set of best plans which are individually generated by invoking the query optimizer separately for different points in the selectivity space. By contrast, the *JoinEnumerationViewer* system analyzes one full search space produced by a single optimization process which includes also all generated physical subplans rejected during that process. To our knowledge, there are no other published research or industrial systems for visualizing and analyzing search spaces generated by different join enumeration algorithms.

II. Description of the System

The *JoinEnumerationViewer* is implemented as a Java application, with the GUI written with Swing classes. It uses a customized renderer for logical and physical plan graphs visualisation and a customized Java SAX parser for parsing and generating the internal structures.

The join enumeration algorithms supported in the SQL Anywhere Optimizer are all partition-based algorithms. A partition-based algorithm enumerates logical partitions of the form (S_1, S_2), corresponding to the logical join $(S_1) \bowtie (S_2)$, for a subset $S = S_1 \cup S_2, S \subseteq V$ (V is the set of vertices of the query hypergraph). These algorithms have the common characteristics that a logical partition is enumerated, and then physical plans corresponding to that partition are generated and costed. The best physical plans seen so far for a certain subset S are saved in the memoization table (referred to as $mTable$). The pruning strategies, in general, are applied only to the generated plans, not to the enumerated partitions, by comparing their costs with both the best complete plan cost generated so far and with the best plan cost for S. Another common feature of these algorithms is that the partitions for a subset S are enumerated in random order, at very different times during enumeration. For example, for a clique query of size 10, *DPhyp* [7] enumerates 28,501 partitions, and the partitions for the same subset of size 7 are enumerated at positions 9508, 9912, 10371, 12291, 12399, 13222, etc.

The Optimization Log: The optimization log is a compact representation of the optimizer search space where each line in the log represents a physical subplan (e.g., a join node, a leaf node, or a unary node (e.g., groupby nodes)) generated during optimization. Each log entry has a unique id and describes physical and logical properties of the costed subplan. A costed subplan node entry has, in general, the following format:
$ID.../ID_{child_1}/.../ID_{child_n}/predicates/properties$
containing its children ids, its logical and physical properties, pruning decisions applied to this plan, etc. Its position in the optimization log indicates the order in which the physical subplan was generated and costed. The optimization log also contains global statistics about memory consumption and optimization time breakdown.

The Logical Plan Graph: The *JoinEnumerationViewer* system translates the optimization log into two internal structures, namely the *Logical Plan Graph* and the *Physical Plan Graph*. From the optimization log, the *JoinEnumerationViewer* system infers all the enumerated partitions of the form $(S_1, S_2), S = S_1 \cup S_2, S \subseteq V$. The *Logical Plan Graph* has the leaf nodes all subsets $S \subseteq V$, while the internal nodes are all subsets S_i which are part of at least one enumerated partition. For a partition $(S_1, S_2), S = S_1 \cup S_2$, two edges are defined between the leaf node S and the internal nodes S_1 and S_2, respectively. The *Logical Plan Graph* doesn't preserve the order in which the partitions were enumerated. Its main goal is to allow a difference graph algorithm to find common or missing partitions enumerated by different join enumeration algorithms. Fig. 1 shows a comparison of two *Logical Plan Graphs* generated from optimization logs of the *DPhyp* algorithm and the *MinCutHyp* algorithm for a 5-way star join query. The partitions enumerated only by one enumeration algorithm are highlighted in blue and red, respectively, while the common partitions are left unmarked. Note that *MinCutHyp* algorithm enumerates unconnected partitions (i.e., cross-products) highlighted in red in Fig. 1, while the *DPhyp* algorithm exhaustively enumerates

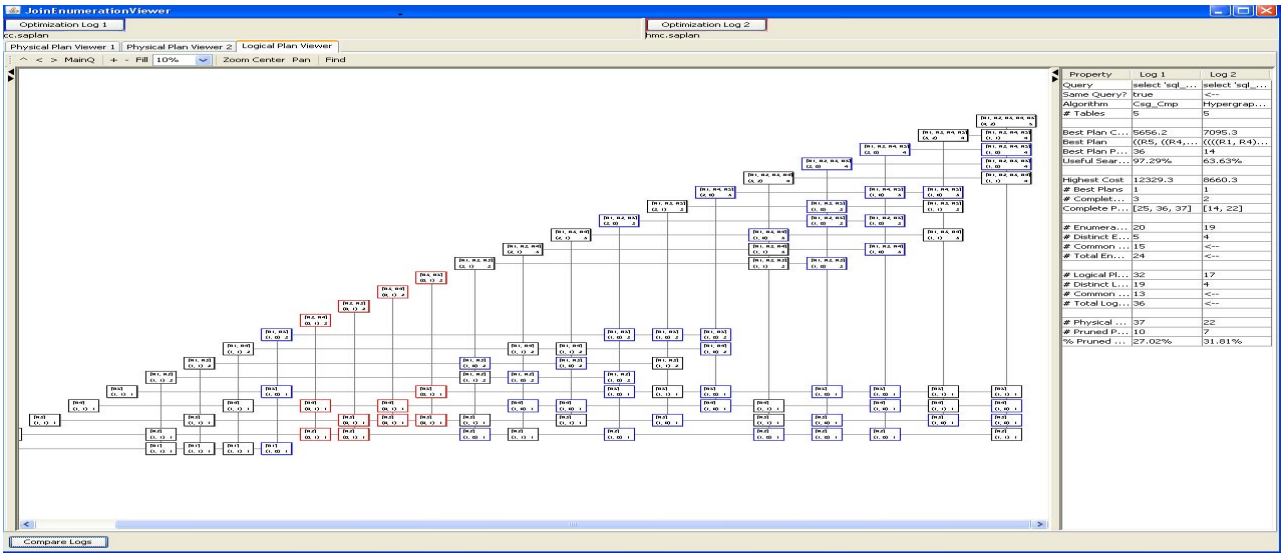

Fig. 1. *JoinEnumerationViewer*: Logical Plan Graphs comparison of search spaces generated by the *DPhyp* algorithm and the *MinCutHyp* algorithm for a 5-way star join query

all connected partitions.

The Physical Plan Graph: This graph is a directed ordered graph of the generated physical plans. Each internal node corresponds to a generated physical join operator or a physical unary operator. It has at most two children corresponding to the children of the physical operator. Unlike the *Logical Plan Graph*, the *Physical Plan Graph* preserves the order in which the physical plans were generated during optimization. The visualization of the *Physical Plan Graph* has the height (the Y axis) of each node set to the total estimated cost of that subplan. The order of the nodes (the X axis) corresponds to the order in which the plans were generated by the join enumeration algorithm.

The resource consumption statistics: The optimizer currently can record two types of resource statistics: (1) buffer pool utilization of the optimization process; and (2) times spent for different activities during join enumeration. Comparing these types of statistics for different enumeration algorithms may reveal unexpected details of how efficient the algorithms are, or where the effort should be spent for improving an algorithm's efficiency. We list here some of the breakdown times of the optimization process recorded in the optimization log:

Initializing Optimization Time: the time spent to set up the initial structures used during join enumeration which includes building the enumerator's representation of the query graph (common to all the algorithms), and other structures specific to each algorithm.

Initial backtracking Time: the time spent running the cheap *backtracking* algorithm to get an initial upper bound for the estimated cost.

Costing Physical Plans Time: the time spent costing generated physical plans corresponding to enumerated partitions. Note that not all enumerated partitions have generated physical plans for some join enumeration algorithms such as *ordered-DPhyp* [2].

Enumeration Time: the time spent enumerating logical partitions.

XXX mTable: the time spent working with memoization table (e.g. lookups, inserts) during the XXX activity. XXX can be, for example, the logical partition enumeration, generating physical plans, costing.

Fig. 2 depicts an example of the optimization time breakdown for a 10-way star join query for three algorithms *DPhyp*, *ordered-DPhyp*, and *TopDown*.

III. Demonstration Scenarios

In the demonstration scenarios, the user may interact with the demonstration process in three ways: (1) by choosing an enumeration algorithm and its options; (2) by choosing among sets of provided SQL statements and (3) by using directly the *JoinEnumerationViewer* to focus on some specific features, such as navigation of a search space, visualizing best plans, etc. The *JoinEnumerationViewer* system will then allow (A) the analysis of the differences between Logical Plan Graphs generated from optimization logs from different join enumeration algorithms; (B) the analysis of the differences between Physical Plan Graphs generated from optimization logs from different join enumeration algorithms; (C) the analysis of the resource consumption recorded in the optimization logs generated by different join enumeration algorithms.

The available join enumeration algorithms implemented in the SQL Anywhere Optimizer are:

- The *backtracking* algorithm. This algorithm enumerates left-deep trees and employs heuristic pruning strategies based on the ordering of the next candidate arrays. It uses no memoization during enumeration process, hence it can

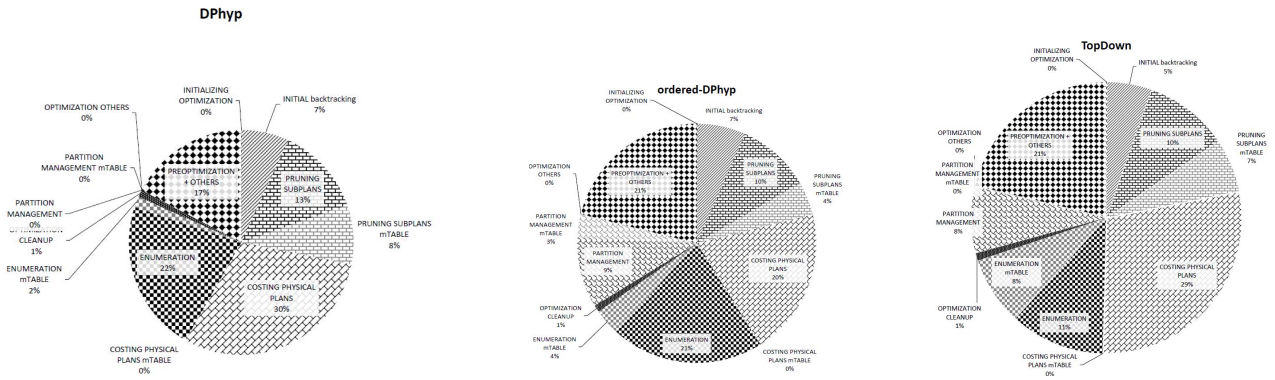

Fig. 2. Optimization time breakdown for 10-way star join query

be used when system resources are limited, for example for a mobile device. This algorithm is used for obtaining, with limited amount of resources, an initial complete plan whose estimated cost is used for cost-based pruning when a new enumeration algorithm is invoked.

- The *backtrackingM* algorithm. This algorithm is an adaptation of the *backtracking* algorithm when the physical and logical properties of a partial access plan are saved in a memoization table and used for cost-based pruning when the same table expression is enumerated.
- The *DPhyp* algorithm. This algorithm is a dynamic-programming algorithm which enumerates bushy-trees. It is adapted from Moerkotte and Neumann algorithm described in [7].
- The *ordered-DPhyp* algorithm. This algorithm combines *DPhyp* with the *ordered-Par* algorithm described in [2] to enumerate bushy trees. The algorithm adapts its ordering and pruning strategies during the search space generation process based on the complexity and the expensiveness of the table expression being optimized.
- The *parallel-ordered-DPhyp* algorithm. This algorithm parallelizes both the enumeration phase and costing phase of the *ordered-DPhyp* algorithm [2] to speed up the optimization process. The *ordered-DPhyp* algorithm is the only enumeration algorithm which could be easily parallelized due to the clear separation between the enumeration phase of logical partitions and the phase for generating and costing physical plans.
- The *MinCutHyp* algorithm. This algorithm is a top-down with memoization algorithm which enumerates set of partitions for a subset S using the hypergraph min-cut algorithm described in [12]. This algorithm enumerates and costs the partitions in the increasing order imposed by the weights associated to the hyperedges. Very few partitions are enumerated and costed comparing to other algorithms over bushy trees.
- The *TopDown* algorithm. This algorithm enumerates bushy trees in a top-down fashion and it is adapted from DeHaan and Tompa algorithm described in [6].
- The *TopDownBranch* algorithm. This algorithm is adapted from the newly-proposed top-down algorithm by Fender and Moerkotte in [8]. It enumerates bushy trees without cross products in a top-down fashion similar to the *TopDown* algorithm.

REFERENCES

[1] I. T. Bowman, P. Bumbulis, D. Farrar, A. K. Goel, B. Lucier, A. Nica, G. N. Paulley, J. Smirnios, and M. Young-Lai, "SQL Anywhere: A holistic approach to database self-management," in *Proceedings,* IEEE ICDE *Workshops (Self-Managing Database Systems SMDB)*. Istanbul, Turkey, Apr. 2007, pp. 414–423.

[2] A. Nica, "A call for order in search space generation process of query optimization," in *Proceedings,* IEEE ICDE *Workshops (Self-Managing Database Systems SMDB)*. Hanover, Germany, Apr. 2011.

[3] ——, "Incremental maintenance of materialized views with outerjoins," *Information Systems Journal*, vol. 37, no. 3, 2012.

[4] M. L. Kersten, A. Kemper, V. Markl, A. Nica, M. Poess, and K.-U. Sattler, "Tractor pulling on data warehouses," in *Proceedings,* ACM SIGMOD *International Workshop on Testing Database Systems DBTest*. Athens, Greece, Jun. 2011.

[5] G. Moerkotte and T. Neumann, "Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products," in *Proceedings,* VLDB *International Conference on Very Large Data Bases*, 2006, pp. 930–941.

[6] D. DeHaan and F. W. Tompa, "Optimal top-down join enumeration," in *Proceedings,* ACM SIGMOD *International Conference on Management of Data*, Beijing, China, Jun. 2007, pp. 785–796.

[7] G. Moerkotte and T. Neumann, "Dynamic programming strikes back," in *Proceedings,* ACM SIGMOD *International Conference on Management of Data*, 2008, pp. 539–552.

[8] P. Fender and G. Moerkotte, "A new, highly efficient, and easy to implement top-down join enumeration algorithm," in *Proceedings,* IEEE ICDE *International Conference on Data Engineering*, Apr. 2011, pp. 864–875.

[9] I. F. Ilyas, J. Rao, G. M. Lohman, D. Gao, and E. T. Lin, "Estimating compilation time of a query optimizer," in *Proceedings,* ACM SIGMOD *International Conference on Management of Data*, San Diego, California, Jun. 2003, pp. 373–384.

[10] A. Nica, D. S. Brotherston, and D. W. Hillis, "Extreme visualisation of the query optimizer search spaces," in *Proceedings,* ACM SIGMOD *International Conference on Management of Data*, Providence, Rhode Island, Jun. 2009, pp. 1067–1070.

[11] J. R. Haritsa, "The Picasso database query optimizer visualizer," in PVLDB, vol. 3, no. 2, pp. 1517–1520, 2010.

[12] R. Klimmek and F. Wagner, "A simple hypergraph min cut algorithm," Internal Report, Bericht FU Berlin Fachbereich Mathematik und Informatik, Tech. Rep. B 96-02, 1996.