Review of Dynamic Query Optimization Strategies in Distributed Database

Pankti Doshi
PDepartment of Computer Science
Mukesh Patel School of Technology Management and Engineering, NMIMS Deemed-to-be University
ankti.doshi@nmims.edu

Vijay Raisinghani
Department of Information Technology
Mukesh Patel School of Technology Management and Engineering, NMIMS Deemed-to-be University
vijay.raisinghani@nmims.edu

Abstract: **A distributed database is a collection of independent cooperating centralized systems. *Query processing* in a distributed database requires transfer of data from one computer to another through a communication network. Query at a given site might require data from remote sites. In query optimization, a cost is associated with each query execution plan. Cost is the sum of local cost (I/O cost, CPU cost at each site) and the cost of transferring data between sites. The complexity and cost increases with the increasing number of relations in the query. Further, minimizing the amount of data transmission is important to reduce the query processing cost. Due to the large number of parameters affecting query execution cost, a single query can be executed in several different ways. A query execution strategy or plan is required to minimize the cost of query processing. The key problem for query optimization in a distributed database is selection of the most cost effective plan to execute a query. Extensive research has been done for query optimization in distributed databases. Numerous optimization strategies like static, dynamic and randomized strategies have been proposed for determining an optimal plan. However, these optimization strategies require knowledge of the entire system (logical schema, physical schema, availability of resources at other sites etc.), and therefore, are not suitable for an autonomous distributed database system. In an autonomous distributed database system nodes are unaware of each other. We review the traditional optimization strategies, like Static and various dynamic strategies for non-autonomous distributed database systems. We also analyze the suitability of these strategies for autonomous systems. Further, we review *Mariposa*, *Query trading* with DP and IDP and Query Trading with *Processing task Trading* (QTPT) for autonomous distributed database systems. The primary focus of our review is on dynamic optimization strategies.**

Keywords: Database, Distributed Query Optimization, optimization Strategies, Deterministic strategies, Randomized Strategies, Autonomous Systems, Dynamic Programming, Iterative Dynamic Programming, Query Trading

I. Introduction

Distributed systems are a collection of independent cooperating systems, which enables storage of data at geographically dispersed locations, based on the frequency of access by users local to a site. The distributed database also enables combining of data from these dispersed sites by means of queries [4].

The performance of a distributed database query depends on how fast and efficiently data is retrieved from multiple sites. Faster retrieval of data in a distributed database system is a complex problem since multiple sites are involved. Several factors impact the performance of distributed query processing. This factors are selection of appropriate site (when same data is replicated at multiple sites), order of operation (like select, project and join) and selection of join method (like semi join, natural join, equi join etc.). Due to the large number of factors involved, there could be multiple execution plans for a single query. Each plan is associated with a cost and the objective of a distributed *query optimizer* is to find a plan with lowest possible cost (*optimal plan*). The execution cost is expressed as a sum of I/O, CPU and communication cost [1].

Query optimization in a distributed database involves phases like global optimization and local optimization [2] [5]. In a distributed database global optimization is an important phase as multiple sites are involved.

Query optimization involves two main tasks, *search space* generation and finding an optimal plan from the search space, using *search strategies* and *cost model*. The search space is a set of alternative execution plans for the input query. A given query is represented as query trees (Join trees) in a search space using transformation rules. If a given query involves many operators and many relations, then the search space will become very large, because it will contain a large number of equivalent query trees for the given query. Investigating a large search space may increase optimization time and cost. Hence, query optimization imposes some restriction on the size of the search space. To reduce the size of the search space, a restriction is imposed on the shape of a query tree. Several classes of join trees like linear and bushy trees [3] exist. The problem of finding an optimal join tree from set of alternatives (search space) is solved by using several search strategies. The search strategy is used to explore the search space in order to find an optimal plan from set of alternatives using a cost model [5].

Query optimization could be static or dynamic. In this paper, we review static and several dynamic optimization strategies for autonomous and non-autonomous distributed database system. In a static optimization strategy, a given query is parsed, validated, optimized once and stored in the database. Dynamic optimization strategy works in bottom-up way by building more complex plans from the simpler plans. System R [9] query optimizer is based on dynamic strategy.

978-1-4244-8679-3/11/$26.00 ©2011 IEEE

The memory requirement in dynamic optimization strategy increases as the number of relations in the query increases. This disadvantage of dynamic optimization strategy is overcome by randomized optimization strategy, which performs random walk in a search space. These strategies are suitable for non-autonomous distributed database systems, since they require complete system knowledge (physical schema, logical schema etc.). However, in an autonomous distributed database system, nodes do not have complete system knowledge. Hence, new techniques like Mariposa [17] and Query Trading [16] are useful for autonomous distributed database system.

The rest of the paper is organized as follows. Section 2 reviews optimization strategies for non-autonomous distributed database systems. Section 3 reviews several dynamic strategies for an autonomous environment. Conclusion and future work is presented in section 5.

II. Query optimization strategies for non-autonomous distributed database systems

In this section we review the existing literature for query optimization in non-autonomous distributed database systems. Query optimization could be done at compile time or run time. The traditional approach is to compile and optimize query before actual execution and store it in database. Another approach is to dynamically generate and choose an optimal plan while execution. Below we discuss several approaches for query optimization for non-autonomous distributed database systems.

1. *Static approach:* The most naïve approach is *static optimization strategy*. In this approach, a given query is parsed, validated and optimized once. Plans for these queries are stored in the database, and then retrieved, and executed whenever the application program is executed. Static optimization is widely used in centralized systems. SDD1 [7] and R* [8] for centralized systems use this approach. Static optimization approach has certain limitations. The assumptions made at the time a query is submitted will rarely hold throughout the duration of query processing, due to variations in execution environment. This execution environment variations increase in a distributed environment, as multiple sites and network communication between them is involved. As a result, static query optimization and execution technique is ineffective in distributed environment. Dynamic optimization strategy helps to overcome these shortcomings.

2. *Dynamic optimization strategy (deterministic strategy*) works in bottom-up way by building more complex plans from simpler plans [6] as shown if figure 1. Here equivalent partial plans are constructed and compared on some cost model [2]. To reduce the optimization cost, partial plans that are not likely to lead to optimal plan are pruned as soon as possible, and cheaper plans are used to construct the full plan [2]. The most popular dynamic strategy is *dynamic programming***.** Dynamic programming was pioneered by IBM's System R database project [9].

Step	Plans Generated
1	{A} {B} {C} {D}
2	{AB} {AC} {AD} {BC} {BD} {CD}
3	{ABC} {ABD} {ACD} {BCD}
4	{ABCD}

Dynamic Programming Algorithm

Step 1: Build an access plan for every relation involved in the query.
Step 2: Prune all inferior plans of step 1 and only keep plans close to optimal.
Step 3: Enumerate all two-way join plans using the Step 2 access plans as building blocks at each relevant site
Step 4: Prune all inferior plans of step 3 and only keep plans close to optimal.
:
:
Step n: Build all n-way join plans.
Step n+1: Prune all inferior plans of step n and only keep plans close to optimal.

Figure 1: Steps for dynamic programming algorithm. A, B, C, D are relations. Generated plans are shown; pruning is not shown.

As shown in figure 1, in the first step algorithm builds an access plan for every table involved in a query. Using basic access plan of first step, algorithm will enumerate all two-way join plans for all relevant sites. The algorithm continues in this way until it has enumerated all the *n*-way join plans in query. Dynamic programming will discard inferior plans as early as possible, if some alternative plan exists that does the same work [9]. The advantage of dynamic programming is that it produces one or even several equivalent optimal left deep trees. However, dynamic programming algorithm has high memory consumption and exponential time complexity, with increasing number of relations, for generating and storing all partial plans found while joining tables. Though the algorithm has exponential time complexity, it finishes in reasonable time for joins involving a small number of relations, and it is guaranteed to find the optimal plan for executing the query, however its performance degrades as the number of relations increases beyond 10-15 in the query [10]. Below we present Iterative Dynamic Programming [10] and Randomized optimization strategy [12] which overcomes this disadvantage.

3. *Iterative Dynamic Programming (IDP) [10]* Dynamic programming algorithm produces good optimization plans, however *its* high complexity can be prohibitive if complex queries need to be processed [10]. Donald Kossmann and Konard Stocker in their work [10] proposed a new optimization technique *iterative dynamic programming* (IDP), which is a combination of both dynamic programming and *greedy* algorithms. Greedy algorithm like dynamic programming constructs complex plans from simple plans in a bottom-up way. However, in

the second phase greedy algorithm carries out a very simple and rigorous selection of the join order. IDP is able to adapt in case there are not enough resources (example, shortage of memory) available. To understand how IDP works, let us assume that the system has enough memory to store all access plans, two-way, three-way, *k*-way join plans for a query with *n* tables, but no more than *k* plans (assume n > k). If dynamic programming is used, when the system starts generating the *(k+1)*-way join plan, it will crash due to insufficient memory. IDP generates all two-way, three-way, *k*-way join plan, same like dynamic programming. When constructing *(k+1)*-way join plan, IDP selects one of the *k*-way join plan and discards all other access and join plans, that involve one of the tables of the selected plan. IDP then restarts in order to build (*k+1*)-way, *(k+2)*-way and so on join plans [10]. The authors have shown that IDP produces better plans than any other algorithm [10].

4. *Randomized optimization strategies [12]* built one or more start plans by a greedy strategy, and then it tries to improve the start plan by randomly visiting its other nodes. Randomized optimization strategies include *simulated annealing* [12] and *iterative improvement* [14] techniques. Randomized algorithms avoid high cost of optimization in terms of memory and time consumption [11]. Experiments show that randomized optimization strategy provides better performance than other optimization strategies when the number of relations involved in the query increases beyond 10 to 15 [12]. The running time for randomized algorithms cannot be predicted. Further, randomized algorithms do not guarantee an optimal plan.

Above, we presented an overview of various optimization techniques for autonomous systems. Next, we discuss the applicability of these techniques to autonomous distributed database systems.

A. Analysis of optimization strategies in view of non autonomous distributed database systems

Dynamic programming produces most optimal plans; however, iterative dynamic programming produces the best plans in situations in which dynamic programming is not viable for its high memory consumption. Dynamic query optimization techniques have certain assumptions like 1. Static data allocation: objects like relations, records etc. cannot easily change sites to reflect changing access pattern, 2. Uniformity: all the sites have the same processing capabilities and memory. In non-autonomous distributed database systems, the query optimizer decomposes the query and decides where to execute each of the sub queries [15]. This is possible because the query optimizer assumes that each node has same processing capabilities.

In an autonomous distributed database system, nodes do not have complete knowledge of the system. The nodes are independent to execute a query or reject execution of query based on its processing capabilities and current load. Hence, before actual execution of the query, a node where query was initiated should ideally identify all other nodes which can participate. Run time conditions, such as the existing load on the site and communication costs, would have a significant impact on the execution cost of a query. It is evident from the discussion above, that optimization strategies for non-autonomous distributed database systems are not suitable for autonomous distributed database systems.

In the next section we describe query optimization strategies for autonomous distributed database systems, which address the challenges of an autonomous distributed database system.

III. Query Optimization strategies for autonomous distributed DBMS

In this section we present, query optimization techniques for autonomous distributed DBMS. Here, we discuss *Mariposa* algorithm [15], *Query Trading* (QT) algorithm [16] and variations of query trading algorithm.

In an autonomous distributed database management system, all nodes are independent and unaware of physical schema, logical schema, data statistics etc. Here, the optimizer does not select the nodes which will participate in query execution. Nodes independently decide whether to participate or not. Hence, before actual query execution, all participating nodes are identified. A node where query is initiated (initiator node) will request all other nodes for query execution. All other nodes will identify part of a query that they can execute and estimates its total cost including disk access cost and communication cost based on its processing capabilities and current system load. All nodes will send a reply to the initiator, identifying the part of query and cost. Based on these replies, the initiator will select the nodes with lowest possible cost and declare the selected nodes as *winners*. The query plan generator in the initiator will develop a query execution plan based on these winning nodes.

Mariposa and Query Trading are based on this concept. The advantage of this concept is that each site has local autonomy to determine how much an operation costs, and can take into account factors such as resource consumption, response time and current system load [17]. The terminology used by Mariposa and Query Trading algorithm is listed below [16]:

- *Buyer node*: Initiator node, where query is initiated. It is assumed that the buyer node does not have sufficient resources to answer the query.
- *Seller node*: Nodes that are capable of providing data relevant to query provided by buyer nodes
- *Request for bids* (RFB): Buyer nodes will send RFBs to seller nodes, asking cost of sub-query that seller node can execute

Based on this terminology architecture used by Mariposa and Query Trading algorithm is as shown in figure2.

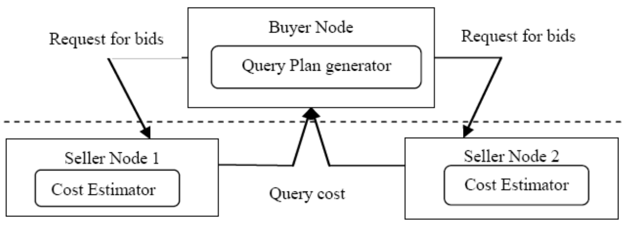

Figure 2: Architecture for Query Trading and Mariposa

Below we discuss Mariposa and query trading algorithm.

1. Mariposa [15]: In Mariposa, queries are submitted by a buyer node. A query starts with a budget. Once budget is decided, query is parsed and given to a single site optimizer. Single site optimizer optimizes query as if data is not fragmented and prepares a plan. This single site plan is further converted into fragmented plan by fragmenter according to number of fragments in query. The buyer will collect fragmented query plans prepared by fragmenter and advertise for bids to various sites. After collecting bids from various interested sites, buyer decides which one to accept and notifies winning sites by sending a bid acceptance [15]. Each Mariposa site is free to bid or reject, hence it has total local autonomy. Mariposa generates optimal plans and is suitable for autonomous distributed database management system. However, it does not support fully autonomous environment as it needs data statistics, indices information and partitioning information for generating good quality of plans. This disadvantage of Mariposa is overcome by Query Trading algorithm.

2. Query trading (QT) algorithm [16]: Fragkiskos Pentaris and Yannis Ioannidis proposed Query Trading which requires lesser information from distant nodes, as compared to Mariposa, allowing higher node autonomy. The basic idea of QT algorithm is to consider queries and query answers as commodities and the query optimization procedure as a trading of query-answers between nodes.

The query trading algorithm includes buyer side algorithm and the seller side algorithm. The buyer asks seller for assistance in evaluating some queries. The seller nodes will rewrite query according to fragment of data that they have and using local optimizer will generate partial query execution plan. The sellers make offers that contain cost of the answer of the queries and processing tasks involved in solving query. Based on seller's bids, buyer decides winner with lowest bid. The query plan generator at buyer will combine the queries that won the bidding procedure to build possible execution plans for the original query. Local optimizer at seller node and query plan generator at buyer node uses dynamic programming algorithm or IDP to search optimal plan. The authors have also proposed the use of improvisations of dynamic programming and iterative dynamic programming (IDP *(k)* and IDP-M *(k, m)* [17]) for use with query trading algorithm. They call these approaches as QT-DP and QT-IDP. We briefly describe IDP *(k)* and IDP-M *(k, m)*. IDP (k) enumerates all *k*-way joins, finds cost of all *k*-way joins contacting sellers using a single round of communication. Like IDP it chooses one plan out of all sub-plans using an evaluation function and throws away all other plans. IDP-M works like IDP and IDP *(k)*, but instead of selecting one plan and throwing other plans, it selects *m* plans out of *k* and throws away other plans. IDP-M *(k, m)*, produces a more optimal plan as compared to IDP or IDP *(k)* [17]. As compared to IDP and IDP-M, IDP suffers from possibility of discarding a potential optimal plan, in the pruning stage.

In a distributed query optimization, an important factor affecting the overall performance of distributed execution plans is the selection of nodes that will eventually process the data. The processing can be performed either at seller nodes or buyer nodes. In a query trading, the seller only processes data that is locally available, while the buyer performs all left-over processing on the data received from the sellers. These restrictions may lead to non-optimal plans, especially when the buyer is overloaded [17]. Hence, to handle such situation a variant of query trading algorithm, query trading with final step of processing task-trading (QTPT) is proposed in [16].

3. Query trading with processing task trading: QTPT is an extension of query trading. It works in two phases. First, it runs the normal query trading algorithm to find an initial distributed query execution plan and then in second phase again it asks for bids from seller nodes for all processing tasks involved in plan. Based on the bids, the tasks would be distributed to the winning nodes. QTPT produces better plans compare to QT, however the times required for optimization increases due to an additional phase.

In the next section, we analyze the optimization strategies for autonomous distributed systems.

A. Analysis of optimization strategies for autonomous distributed database system

Research show that Mariposa produces less efficient plans compared to Query Trading (QT) algorithm [16].

Mariposa requires more information for query optimization than QT. In autonomous systems, to increase local autonomy the optimizer consults the data sources involved in an operation to find the cost of that operation. Hence the dominant cost in optimization becomes the cost of contacting the underlying data sources. Below we compare the various optimization algorithms proposed for autonomous distributed database systems on parameters like time delay, startup cost and information required.

- **Time Delay:** Experiments [16] show that QT with PT algorithms substantially improves the plans produced by QT-IDP. The disadvantage of PT algorithm is it spends a lot of its time idle, waiting for replies to their bids concerning queries and processing tasks. Hence, network and seller processing delay times are a large fraction of the total optimization time [16].
- **Startup cost:** The execution time of the algorithms exponentially depends on the number of joins. The QT-IDP algorithm has a start-up cost of a single round of message exchanges caused by the bidding procedure. The QTPT algorithm has a start up cost of two rounds of

message exchange, one for finding initial query execution plan and second for requesting bids for processing tasks in query being optimized [16].

- **Types of information required by algorithms:** Algorithm that requires less information is more suitable for the autonomous distributed database system. Table 1, shows the amount of information required by each algorithm.

	QT/QT-IDP	Mariposa	DP/IDP
Logical Schema	Required	Required	Required
Physical Schema – Partitioning		Required	Required
Physical Schema- Indices		Optional	Required
Data Statistics		Required	Required
Workload			Required

Table 1: Amount of information required by algorithms

In the sections above we discussed various optimization algorithms proposed for non-autonomous and autonomous distributed database systems. In the next section, we conclude the review, and provide pointers for future research work.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we have discussed the various approaches to distributed query optimization for autonomous and non-autonomous distributed database system. We discussed strategies like static, dynamic and randomized. We discussed dynamic strategies like dynamic programming and iterative dynamic programming and discussed how this basic approach is unsuitable for autonomous environment. Two new dynamic approaches like Mariposa and Query Trading, which use variants of DP and IDP, were discussed for autonomous environment. In a non-autonomous environment DP and IDP produce the best plans, but require complete information about other nodes in the system. Research shows that Query Trading with IDP works better in autonomous environment compared to Query Trading with DP. Query Trading with Process Trading is an improvement over QT; however it requires an additional phase of bidding for processing tasks.

Future Work: There are short comings of Query Trading and Query Trading with Process Trading which requires further investigation. Some of the short comings are: 1. A large number of messages are transferred between nodes in QT and QTPT. This needs to be minimized. 2. The query initiator node wastes time waiting for responses from other seller nodes. These impacts the overall time required for optimization time, and hence needs to be minimized.

REFERENCES

[1] OSZU, M.T and Valduriez P., "Principles of Distributed database systems", Prentice Hall International, NJ, 1999.

[2] Alaa Aljanaby, Emad Abuelrub and Mohammed Odeh, "A Survey of Distributed Query Optimization", The international Arab journal of Information Technology, Vol.2, No.1, January 2005.

[3] Ray C, "Distributed Database systems", Pearson Publication, 2009

[4] Barry E. Jacobs, Cynthia A. Walczak, " Optimization algorithms for distributed queries", IEEE transactions on software engineering, Vol. SE-9, No.1, January-1983.

[5] Reza Ghaemi, Amin Milani Fard, Hamid Tabatabee and Mohid Sadeghizadeh, "Evolutionary query optimization for heterogeneous database systems", World academy of science, Engineering and Technology, 43, 2008.

[6] Kossmann D, "The state of art in distributed query processing", ACM Computing surveys, Vol.32, No.4, December 2000, PP 422-469.

[7] Bernstein, P., Goodman, N., Wong, E., Reeve, C., and Rothnie, J.1981, "Query Processing in a system for distributed databases (SDD-1)", ACM Transactions on Database systems 6, 4 (Dec), PP 602-625.

[8] L.M.Hass, "R* : A research project on distributed relational DBMS", Database Engineering, Vol.5, 1982

[9] 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" in proceedings of the ACM SIGMOD conference on management of data, Boston, USA, PP 23-24 1979.

[10] Donald Kossmann and Konard Stocker, "Iterative dynamic programming: A new class of query optimization algorithms", ACM Transactions on database systems, Vol.25, No.1, March 2000.

[11] ONO, K. and Lohman, G, 1990, "Measuring complexity of join enumeration in query optimization", in proceedings of the 16th international conference on very large databases (VLDB, Brisbane, Australia, Aug.), VLDB Endowment, Berkley, CA 314-325.

[12] Yannis E. Ioannidis and Younkyung Cha Kang, " Randomized algorithms for optimizing large join queries", in Proceedings of the ACM SIGMOD Conference on Management of Data, Atlantic City, USA, pp 312-321, 1990

[13] S Kirkpatrick, C D Gelatt, Jr, and M P Vecchi "Optimization by Simulated Annealing", science 220, 4598, May 1983, PP 671-680.

[14] S Nahar, S Sahni and E Shragowitz "Simulated Annealing and Combinatorial Optimization", in proceedings of the 23rd Design automation conference,1986,pp93-299.

[15] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Stalien, Andrew Yu "Mariposa: a wide area distributed database system" The VLDB journal, 1996, P:48-63.

[16] Pentaris F and Ioannidis Y "Query Optimization in Distributed Networks of Autonomous Database Systems", ACM Transactions on Database Systems, Vol. 31, No.2, June 2006, P: 537-583.

[17] Amol V. Deshpande and Joseph M. Hellerstein, "Decoupled Query Optimization for Federated Database Systems", in proceedings of the 18th International Conference of Data Engineering (San Jose, CA). IEEE Computer Society, Los Alamitos, P:716-79