World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:12, No:3, 2018

A Query Optimization Strategy for Autonomous Distributed Database Systems

Dina K. Badawy, Dina M. Ibrahim, Alsayed A. Sallam

***Abstract*—**Distributed database is a collection of logically related databases that cooperate in a transparent manner. Query processing uses a communication network for transmitting data between sites. It refers to one of the challenges in the database world. The development of sophisticated query optimization technology is the reason for the commercial success of database systems, which complexity and cost increase with increasing number of relations in the query. Mariposa, query trading and query trading with processing task-trading strategies developed for autonomous distributed database systems, but they cause high optimization cost because of involvement of all nodes in generating an optimal plan. In this paper, we proposed a modification on the autonomous strategy K-QTPT that make the seller's nodes with the lowest cost have gradually high priorities to reduce the optimization time. We implement our proposed strategy and present the results and analysis based on those results.

***Keywords*—**Autonomous strategies, distributed database systems, high priority, query optimization.

I. Introduction

QUERY processing includes translation of high-level queries into low-level expressions that can used at the physical level of the file system.

Query Optimization is the process of finding the best strategy in order to execute the given query from a set of alternatives [1], [2]. Query optimization and actual execution of the query needed to get the result consists of three steps: parsing and translation, optimization and execution of the query submitted by the user; these steps shown in Fig. 1 [2]. A relational algebra operations and communication primitives like send or receive operations describe a distributed query execution strategy to transfer data between sites [3].

The query optimizer that follows this approach consists of three components: A search space, a search strategy and a cost model [3], [4]. The search space is the collection of different execution for performing the input query. The search strategies are equivalent, in the sense that they produce the same result but they differ on the execution order of operations and the way these operations are implemented [5], [6]. The search strategy inspects the search space in order to choose the best plan. It determines how to test the plans and how to arrange them [5]. The cost model predicts the cost of a given execution plan which may consist of the following components [6].

1. Secondary storage cost: It is the cost of searching for reading and writing data blocks on secondary storage.
2. Memory storage cost: This cost related to the number of memory buffers needed during query execution.
3. Computation cost: It is the cost for performing in memory operations on the data buffers during query optimization.
4. Communication cost: This cost responsible for shipping the query and its results from the database site to the site or terminal where the query originated.

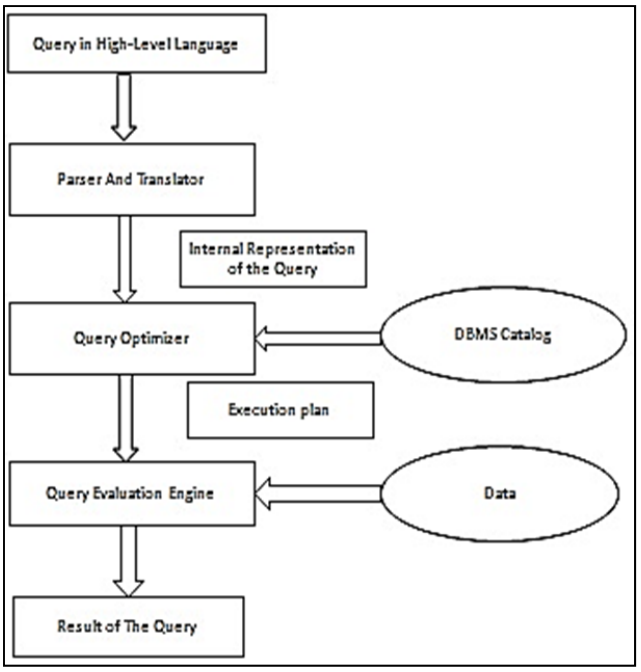


Fig. 1 Query processing steps

Homogeneous distributed database systems strategies divided to non-autonomous and autonomous strategies. For non-autonomous strategies, all nodes are aware of physical schema, logical schema, and data statistics such as deterministic and randomized strategies with their types. For autonomous strategies, all nodes are independent and unaware of each other.

II. Background

There is a central optimizer, which does not support total node autonomy, a non-autonomous distributed database

Dina K. Badawy is with Computers and Control Engineering Dept. Tanta University, Egypt (e-mail: dina.khattab@f-eng.tanta.edu.eg).

Dina M. Ibrahim (Lecturer) is with Computers and Control Engineering Dept. Tanta University, Egypt (phone: 002-01000250820; e-mail: dina.mahmoud@f-eng.tanta.edu.eg).

Elsayed A. Sallam (Emeritus Professor) is with Computers and Control Engineering Dept. Tanta University, Egypt (phone: 002-01155411019; e-mail: sallam@f-eng.tanta.edu.eg).

system. On the contrary in an autonomous distributed database system, there is no central optimizer where each site has complete control over its resources, we mean that, the participating nodes in query execution independently decide whether to participate or not according to the node's resource capacity and data availability [7], [8]. So, all participating are nodes identified before actual query execution.

Authors in [9] proposed an economic model in order to identify all participating nodes and to support node autonomy in an autonomous system. According to this economic model, as shown in Fig. 2 [10], there are two types of nodes - buyer nodes and seller nodes. The buyer node is the node where a query is initiated (initiator node), where the seller is the node that executes the query.

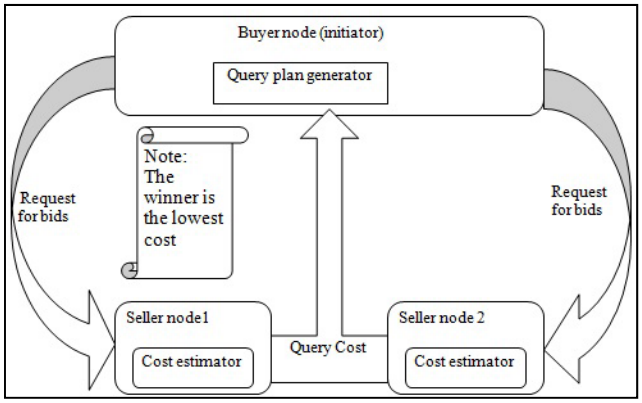


Fig. 2 The Economic model

The Mariposa [11] and Query Trading algorithms [9] depend on this economic model; in general, both algorithms follow the process as stated below:

1. Buyer node prepares Request for bids (RFB) for sub-queries that require cost estimation.
2. Buyer node sends RFB to the seller nodes requesting cost for the sub-query.
3. Seller nodes calculate the costs for sub-queries and send replies back to buyer node.
4. Buyer node, based on replies, decides on an execution plan for the query; if required repeat step 2 and step 3.

III. Earlier Work on Query Optimization Strategies for Autonomous Distributed DBMS

In an autonomous distributed database management system, all nodes are independent and unaware of physical schema, logical schema, and data statistics.

In this paper, we discuss the query optimization strategies for an autonomous Distributed Database Management System (DDBMS). These strategies are the Mariposa strategy, Query Trading (QT) strategy, and variations of the Query Trading strategy.

A. Mariposa Strategy

In the Mariposa strategy, a buyer node submit queries, a query starts with a budget that once is decided, the query is parsed and given to a single site optimizer that makes optimization for whole query, as if the data are not fragmented, and prepares a plan [7]. A fragmenter converts a single site plan into a fragmented plan depending on the number of fragments in the query. The fragmented query plans prepared by the fragmenter collect and advertise for bids to various sites; after that, the buyer decides which one to accept, as shown in Fig 3. Each Mariposa site is free to accept or reject, therefore it has total local autonomy [2], [12]. Mariposa generates optimal plans and is suitable for an autonomous distributed database management system. However, it does not support a fully autonomous environment, as it needs data statistics, indices information and partitioning information for generating good quality plans. This disadvantage of Mariposa is solved by the Query Trading strategy [10], [13].

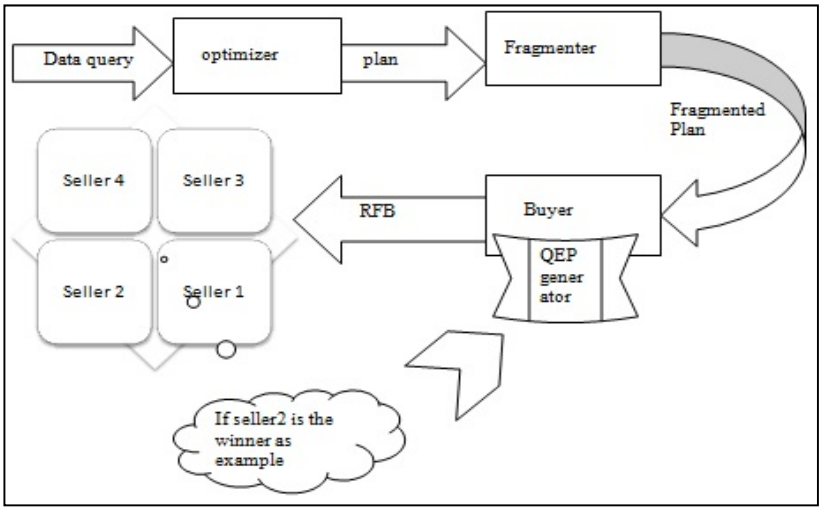


Fig. 3 Architecture of the Mariposa strategy

B. Query Trading (QT) Strategy

Compared to Mariposa, the QT strategy asks reserved nodes for information, but this information is less than the required information, which allows higher node autonomy to participate in executing the queries [7]. QT strategy is considering the queries and query answers as produces and the query optimization procedure as a trading of query answers between nodes.

The query trading strategy means that there are two algorithms, the buyer side algorithm and the seller side algorithm. The buyer sends a request for all sellers asking it for help in evaluating some queries [12]. The seller nodes based on their fragmentation of data will rewrite query and use local optimizer to generate partial query execution plan. They offer that execution including answer's cost of the queries and processing tasks involved in solving query. Buyer decides winner with lowest bid according to seller's bids. Finally, the buyer query plan generator build possible execution plans for the original query by combination of the winner bids [13], as shown in Fig. 4.

In distributed query optimization, the selection of nodes that will eventually process the data is considered as an important factor affecting on the overall performance of distributed execution plans [12], [14]. The processing can either be performed at the seller nodes or at either buyer nodes. In query trading, the seller only processes data that is locally available, while the buyer performs all leftover processing on the data received from the sellers [13]. These restrictions may lead to non-optimal plans, especially when the buyer is overloaded. Hence to handle such a situation, a modified strategy to the QT strategy and the QTPT strategy is proposed [15], [16].

C. Query Trading with Processing Task Trading (QTPT) Strategy

QTPT is an extension of query trading strategy. It works in two phases [7]. In the first phase, it determines the initial distributed query execution plan, while in the second phase it repeats the same process as first phase, and again, sends RFBs for all seller nodes for all processing tasks involved in a plan (i.e., QTPT strategy run the QT strategy twice), as shown in Fig 5. QTPT produces better plans compared to QT; however, the times required for optimization increases due to an additional phase [13].

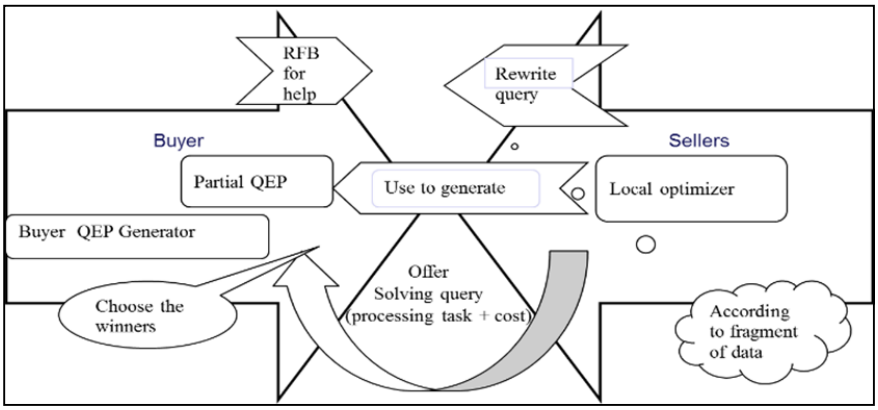


Fig. 4 Architecture of Query Trading

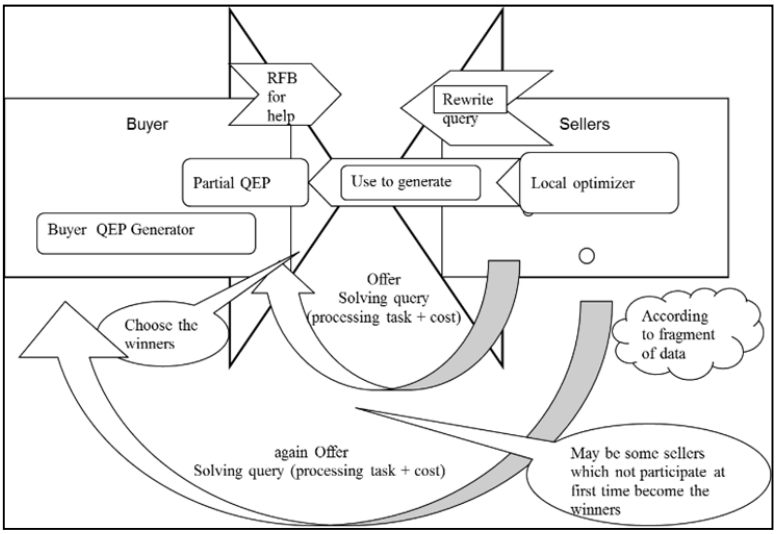


Fig. 5 Architecture of QTPT Strategy

D.K-QTPT Strategy

The K-QTPT strategy works in two phases, such as QTPT [10]. In the first phase, the K-QTPT is typically like QTPT, it determines initial query execution plan [13]. In the second phase, the buyer node will store the K winners from the first phase into a buffering list. Then, the RFBs will start the processing task by requesting from only those K nodes, instead of requesting from all the nodes. This reduces optimization time substantially. Deciding appropriate value of k is one of the challenges for implementing K-QTPT. Fig. 6 shows the architecture of K-QTPT strategy.

In autonomous systems, to increase local autonomy the optimizer consults the data sources involved in an operation to find the cost of that operation [17]. Hence, the main cost in optimization becomes the cost of contacting the underlying data sources; thus, we show that Mariposa produces less efficient plans compared to Query Trading (QT) strategy and requires more information for query optimization than QT strategy [18].

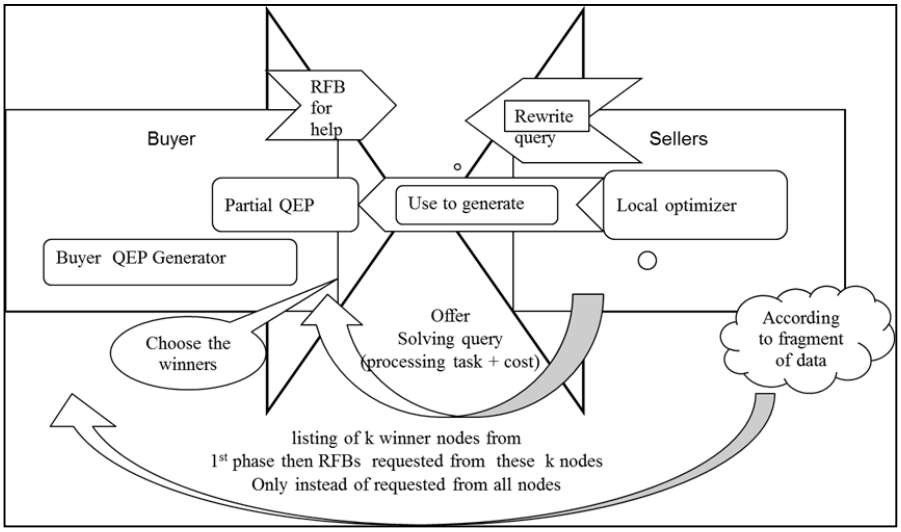


Fig. 6 Architecture of K-QTPT Strategy

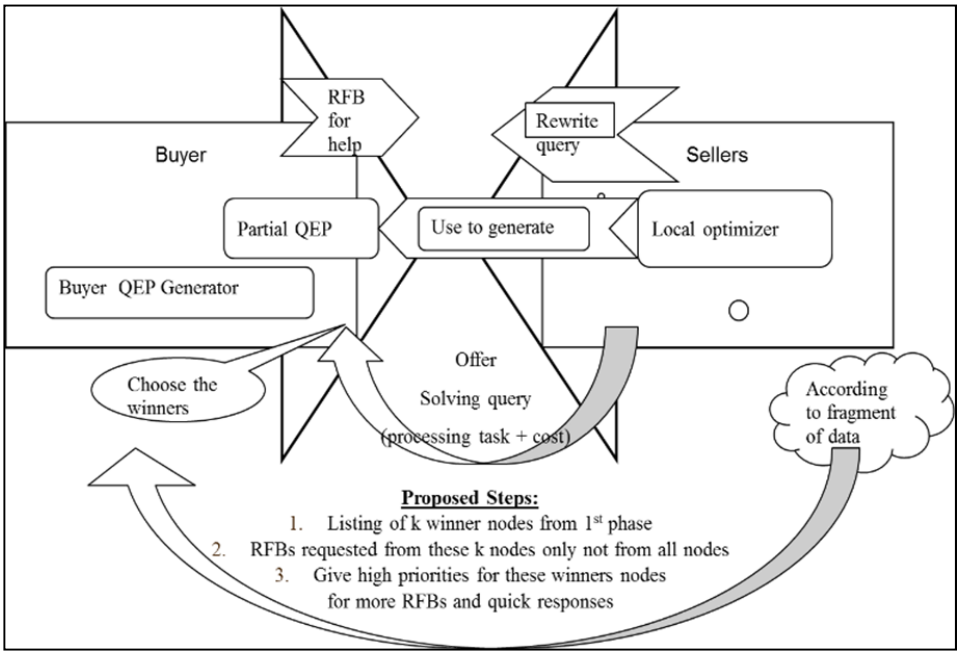


Fig. 7 Architecture of proposed strategy

In the next section, we present our proposed strategy and then we compare it with the various optimization strategies for autonomous distributed database systems based on parameters like time delay and startup cost [19].

IV. The Proposed Strategy

The proposed strategy also works in two phases, like k-QTPT. The first phase is the same as that in k-QTPT strategy,

determines the initial query execution plan (QEP). In the second phase, depending on the winners resulting from the first phase, buyer node buffers a list of K winning nodes then RFBs for the processing task that were requested from only these selected K nodes, instead of requesting from all nodes. After that, the buyer gives high priorities for the winner nodes to increase RFBs and receives quick responses from these winners. This will reduce the optimization time, as shown in Fig. 7. In the next subsection, we present the flowchart, pseudocode, challenges faced, and the experimental evaluation for the proposed strategy, followed by the analysis of the results compared with the QT, QTPT, and K-QTPT strategies.

A. Flowchart

The following flowchart in Fig. 9 explains the steps of our proposed strategy. In the beginning, the user starts entering the query at the initiator node, the buyer node. The buyer sends RFBs for all sellers to ask them for help in evaluating the query execution time. All the seller nodes estimate the query cost and send the response back to the buyer.

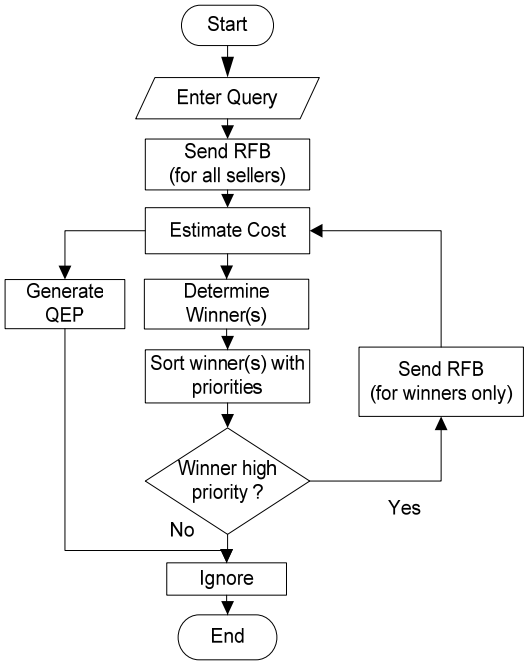


Fig. 8 Flowchart of the proposed strategy

The buyer determines the seller winner nodes that have the lowest cost. Then, sort these winners with priorities; the high priority winner only, which the buyer will send RFB to, and finally produces the QEP to answer the query [20].

B. Pseudocode

In this paper, we propose a modification on the autonomous strategy KQTPT that we call the proposed strategy. The objective of this proposed strategy is to reduce the optimization time taken by KQTPT Strategy. According to our proposed strategy, the buyer sends RFBs to all sellers for estimation of the query cost and every seller replies with the query cost depending on its available resources. The buyer compares the costs to find the seller node with the lowest cost to be the winners. Then, the buyer sorts these winners according to the costs with priorities. The lowest cost of winners takes high priority and gradually completes the winners' priorities. The buyer ignores low priorities and again sends RFBs for high priorities winners. Finally, the buyer generator takes the high priorities winners' costs and produces the QEP that will be close enough to the optimal.

Our proposed strategy starts with implementing a network that consists of one buyer and a number of sellers. The user enters the required query, the buyer asks the sellers to calculate the query execution cost (c). Each seller replies with its c. The buyer collects all the costs and put them ordered in an array list. After that, it compares all costs to find the minimum that is the winner seller node. Once the number of winner nodes k equals two, the buyer compares the two winners and sets high priority for the winner node that has the lowest cost. Then, it displays the cost of the high priority winner from the seller node. After that, the winner sends RFBs for this high priority winner for quick responses. The pseudocode of the proposed strategy is shown in Fig. 9.

Pseudocode
Input: query
Output: query execution plan
- Calculate the query execution costs(c) at each seller node
- Put c in the array list arrList[]
- If c<min
- Set min=c
- End if
- For number of winners k<=2
- If arrList[k] < min then
- Put arrList[] = min & display c of winner with high priority
- End if
- End for

Fig. 9 Pseudocode of the proposed strategy

C. Experimental Setup

We built our experiment starting with designing a DDBMS on a 64-bit workstation device, Core i7 processor and 16 GB of RAM for studying the performance of QT, QTPT, K-QTPT and the proposed algorithm. Our design consists of seven nodes (one buyer and six sellers) interconnected by a network, as shown in Fig 8. All nodes are connected using a LAN with speed of 100 Mbps. We create eight tables using the MYSQL database. Horizontal fragmentation has been done on each table. We have used Java 1.7 and MYSQL 5.1 to simulate the DDBMS [21].

Each node is equipped with MYSQL 5.1. The seven nodes in the emulation setup use Microsoft Windows 7 Ultimate as the operating system. Each node was equipped with NetBeans IDE 8. In order to connect java with MYSQL each node was added mysql-connector-java-5.1.14-bin.jar for the libraries and classpath [22].

D. Challenges Faced

We initially tried to make a LAN network consisting of

three real computers using the Microsoft Windows 7 operating system. Formerly, we faced a problem in the socket port number and we solved it by making different port numbers for the socket and the database. After that, a new problem was found which is the missing of the JDBC driver and solved by adding mysql-connector-java-5.1.14-bin.jar into the classpath and libraries.

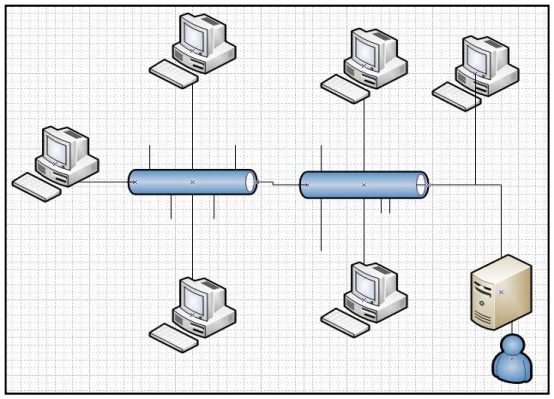


Fig. 10 Organization of the proposed strategy nodes

Then, we met a heap space problem at adding random number of tuples and attributes in db, but it was overcome by setting the privilege of MYSQL. Because the nodes are not multithreading, we programmed a class, which we called socket thread, and the server imports it. Finally, we simulate our work at workstation devices. Based on the above experimental setup, we analyzed the performance of QT, QTPT, k-QTPT strategies and the proposed strategy algorithm.

E. Results and Analysis

In this section, we have measured optimization time, time taken to perform a processing task at query initiating node, buyer node, time taken to perform the processing task at the answering node, seller node, total response time, and total execution time. The main objective of this work is to reduce the optimization time of the proposed algorithm.

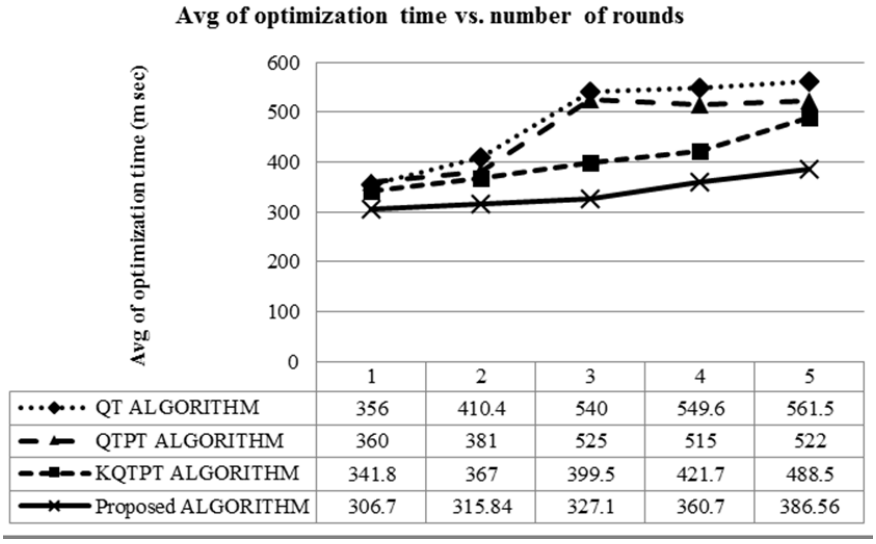
In our experiment, the number of rounds assumed to find the best optimization plan is five, and in each round, we ran the experiment five times and took an average of optimization time (in mille seconds). The results of the average of optimization time comparison of QT, QTPT, K-QTPT and our proposed algorithm is as shown in Fig. 11.

Fig. 11 describes the average of optimization time with the number of rounds. It shows that the optimization time increases with the increase in the number of rounds. The least number of rounds suitable for our simulation is five rounds.

The average of the query execution time at each node with the number of rounds is illustrated in Fig. 12. It shows that the average query execution time increases with the increase of the number of rounds. In addition, the least number of rounds suitable for our simulation is five rounds. The figure shows that our proposed strategy is better than both the QTPT and the KQTPT strategies. However, due to the small load caused by the minimum number of seller nodes, the QT strategy is still the best and produces a QEP that is close and near to the optimal.

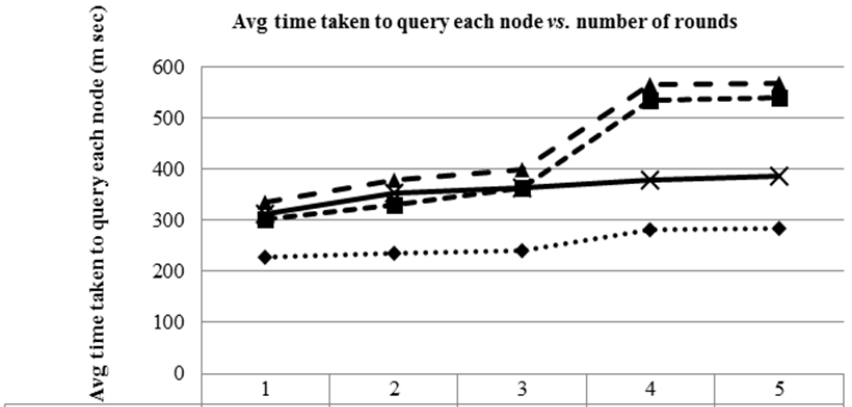
In Fig. 13, the average of the query execution time at each node with the number of seller nodes is explained. It is seen that the average of the query execution time increases with the increase of the number of seller nodes. When we added six seller nodes connected at the buyer, the query execution time at each node was very high. The figure shows that our proposed strategy is better than the QTPT and the KQTPT strategies, but the traditional QT Strategy still gives the best QEP when the buyer was not overloaded.

In our experiment, we avoid the Mariposa strategy from our comparison because it is not support the full node autonomy.



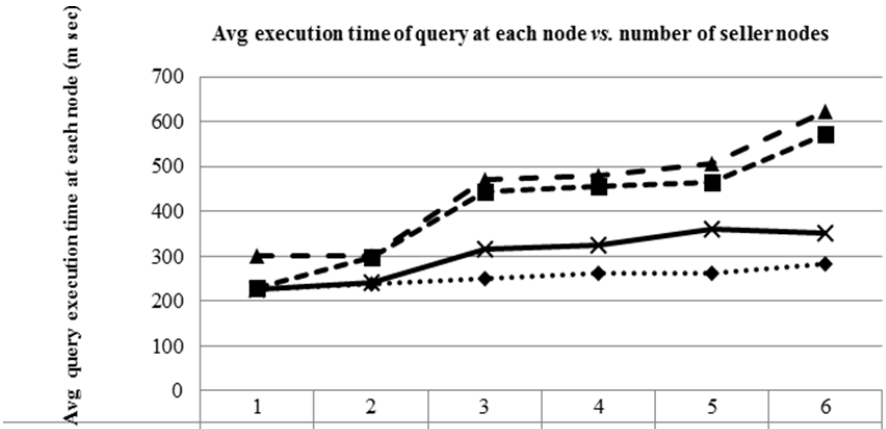
	1	2	3	4	5
QT ALGORITHM	356	410.4	540	549.6	561.5
QTPT ALGORITHM	360	381	525	515	522
KQTPT ALGORITHM	341.8	367	399.5	421.7	488.5
Proposed ALGORITHM	306.7	315.84	327.1	360.7	386.56

Fig. 11 Average of optimization time vs. number of rounds



	1	2	3	4	5
QT	226.67	235.5	241.4	280.7	283.9
QTPT	335	378	398	565	569
KQTPT	301.9	329.2	363.87	533.9	538.8
PROPOSED	312.47	351.93	363.43	379.6	385.37

Fig. 12 Average of execution time at each node vs. number of rounds



	1	2	3	4	5	6
QT	226.44	237.76	249.48	261.4	262.68	284.04
QTPT	301	299	471	478	505	624
KQTPT	228.52	296.16	443	455.52	464.72	573.28
PROPOSED	224.68	239.68	313.92	324.16	360.4	351.52

Fig. 13 Average of query execution time vs. number of seller nodes

V. Conclusion

In this paper, different optimization autonomous strategies are presented. The performance of a distributed database system depends on efficient query optimization. In a non-autonomous environment, the algorithms produce the best plans but require complete information about other nodes in the system. Mariposa, QT, and QTPT are optimization strategies that are proposed to support an autonomous environment. QTPT generates optimal plans compared to QT, but incurs high optimization cost. To reduce the increase of optimization cost, an enhancement of the K-QTPT autonomous strategy is proposed, in which only the K nodes participate in generating the optimal plans. The high priority for the winner seller nodes are chosen to reduce the optimization cost.

Our simulation results demonstrate that the proposed strategy efficiently reduces optimization cost, especially with the increase of the number of rounds, and minimizes the time delay and generates the best plan.

References

[1] E. Ramez, and S. B. Navathe, Fundamentals of database systems, Pearson, 2015.
[2] D. V. Elena, M. Rebollo, and V. Botti, "An overview of search strategies in distributed environments," The Knowledge Engineering Review, vol. 29, no. 3, pp. 281-313, 2014.
[3] Y. E. Ioannidis, "Query Optimization," Computer Sciences Department University of Wisconsin Madison, WI 53706, 2000.
[4] B. M. Alom, F. Henskens, and M. Hannaford, "Query processing and optimization in distributed database systems," *IJCSNS* International Journal of Computer Science and Network Security, vol. 9, no. 9, pp. 143-152, 2009.
[5] M. T. Ozsu, and P. Valduriez. Principles of distributed database systems. Springer Science & Business Media, 2011.
[6] A. Aljanaby , E. Abuelrub , M. Odeh, "A Survey of Distributed Query Optimization," the international Arab Journal of Information Technology, vol. 2, no.1, pp. 48-57, January 2005.
[7] D. Pankti, and V. Raisinghani, "Review of dynamic query optimization

strategies in distributed database," *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*. Vol. 6. IEEE, 2011.
[8] C. Surajit, "An overview of query optimization in relational systems," *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, pp. 34-43,1998.
[9] P. Fragkiskos, and Y. Ioannidis, "Query optimization in distributed networks of autonomous database systems," *ACM Transactions on Database Systems (TODS)*, vol. 31, no. 2, pp. 537-583, 2006.
[10] D. Pankti, and V. Raisinghani, "k-QTPT: A Dynamic Query Optimization Approach for Autonomous Distributed Database Systems," *Advances in Computing, Communication, and Control* 361, pp. 1-13, 2013
[11] D. Amol, and J. M. Hellerstein, "Decoupled query optimization for federated database systems," *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, pp. 716-727, 2002.
[12] T. Robert, "Query Optimization for Distributed Database Systems," Master Thesis University of Oxford, August 2010.
[13] H. Abdelkader, and F. Morvan, "Evolution of query optimization methods," *Transactions on Large-Scale Data-and Knowledge-Centered Systems I*. Springer Berlin Heidelberg, pp. 211-242, 2009.
[14] P. Fragkiskos, and Y. Ioannidis, "Distributed query optimization by query trading," *International Conference on Extending Database Technology*. Springer, Berlin, Heidelberg, pp. 532-550, 2004.
[15] K. Donald, and K. Stocker, "Iterative dynamic programming: a new class of query optimization algorithms," *ACM Transactions on Database Systems (TODS)* vol. 25, no. 1, pp. 43-82, 2000.
[16] K. Donald, "The state of the art in distributed query processing," *ACM Computing Surveys (CSUR)*, vol. 32, no. 4, pp. 422-469, 2000.
[17] Z. Lin, Y. chen, T. Li, and Y. Yu, "The Semi-join Query Optimization in Distributed Database System," *National Conference on Information Technology and Computer Science (CITCS 2012)*, pp. 606-609, 2012.
[18] E. I. Yannis, and Y. Kang, "Randomized algorithms for optimizing large join queries," *ACM Sigmod Record*, ACM vol. 19. no. 2, pp. 312-321, 1990.
[19] K. Stocker ; D. Kossmann ; R. Braumandi ; A. Kemper, "Integrating semi-join-reducers into state-of-the-art query processors," *Data Engineering, Proceedings. 17th International Conference on*. IEEE, pp. 575-584, 2001.
[20] M. Vikash, and V. Singh, "Generating optimal query plans for distributed query processing using teacher-learner based optimization," *Procedia Computer Science*, vol. 54, pp. 281-290, 2015.
[21] S. David, and R. Dantas. *Netbeans IDE 8 Cookbook 2014*. Packt Publishing Ltd, last Access: 2017.
[22] MySQL, A. B. "Mysql 5.1 reference manual, 2009" *Accessible in URL: http://dev.mysql.com/doc*, last Access: 10/10/2017.