

Evolutionary Query Optimization for Heterogeneous Distributed Database Systems

Reza Ghaemi, Amin Milani Fard, Hamid Tabatabaee, and Mahdi Sadeghizadeh

Abstract—Due to new distributed database applications such as huge deductive database systems, the search complexity is constantly increasing and we need better algorithms to speedup traditional relational database queries. An optimal dynamic programming method for such high dimensional queries has the big disadvantage of its exponential order and thus we are interested in semi-optimal but faster approaches. In this work we present a multi-agent based mechanism to meet this demand and also compare the result with some commonly used query optimization algorithms.

Keywords—Information retrieval systems, list fusion methods, document score, multi-agent systems.

I. INTRODUCTION

DISTRIBUTED systems can be taught of as a partnership among independent cooperating centralized systems. Upon this idea number of large scale applications has been investigated during past decades among which distributed information retrieval (DIR) systems gained more popularity due to its high demand. The goal of DIR is to provide a single search interface that provides access to the available databases involving resource descriptions building for each database, choosing which databases to search for particular information, and merging retrieved results into a single result list [1]-[2].

A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. This resource distribution improves performance, reliability, availability and modularity that are inherent in distributed systems. As with traditional centralized databases, distributed database systems (DDBS) must provide an efficient user interface that hides all of the underlying data distribution details of the DDB from the users. The use of a *relational query* allows the user to specify a description of the data that is required without having to know where the data is physically located [3].

Data retrieval from different sites in a DDB is known as *distributed query processing* (DQP). For example, the following query accesses data from the local database as well as the remote sales database. The first table (EMP) found in

site1 and the second table (DEPT) found in site2:

```
SELECT ename, dname
FROM company.emp e, company.dept@sales.goods d
WHERE e.deptno = d.deptno
```

So a *distributed query* is one that selects data from databases located at multiple sites in a network and *distributed processing* performs computations on multiple CPUs to achieve a single result. Query processing is much more difficult in distributed environment than in centralized environment because a large number of parameters affect the performance of distributed queries, relations may be fragmented and/or replicated, and considering many sites to access, query response time may become very high [3].

It is quite evident that the performance of a DDBS is critically dependant upon the ability of the query optimization algorithm to derive efficient query processing strategies. DDBMS query optimization algorithms attempts to reduce the quantity of data transferred. Minimizing the quantity of data transferred is a desirable optimization criterion. The distributed query optimization has several problems relate to the cost model, larger set of queries, optimization cost, and optimization interval.

The goal of DQP is to execute such queries as efficiently as possible in order to minimize the response time that users must wait for answers or the time application programs are delayed. And to minimize the total communication costs associated with a query, to improved throughput via parallel processing, sharing of data and equipment, and modular expansion of data management capacity. In addition, when redundant data is maintained, one also achieves increased data reliability and improved response time.

In this work we propose a multi-agent architecture for distributed query processing. The structure of the paper is as follows. Section II describes an overview of query optimization process. An investigation on related works is declared in Section III, our proposed approach in Section IV, and simulation results in V. We finally concluded the work in the Section VI.

II. QUERY OPTIMIZATION PROCESS

In a relational database all information can be found in a series of tables. A query therefore consists of operations on tables. The most common queries are Select-Project-Join queries. In this paper, we will focus on the join-ordering

Manuscript received January 31, 2008.

R. Ghaemi and H. Tabatabaee are with the CE Department, Islamic Azad University, Quchan branch, Iran (e-mail: rezaghaemi@scientist.com, hamid.tabatabaee@gmail.com).

A. Milani Fard and M. Sadeghizadeh are with the CE Department, Ferdowsi University, Mashhad, Iran (e-mail: milanifard@stu-mail.um.ac.ir, ma_sa638@stu-mail.um.ac.ir).

problem since permutations of the join order have the most important effect on performance of relational queries [4]. The query optimization process shown in Fig. 1, consists of getting a query on n relations and generating the best Query Execution Plan (QEP).

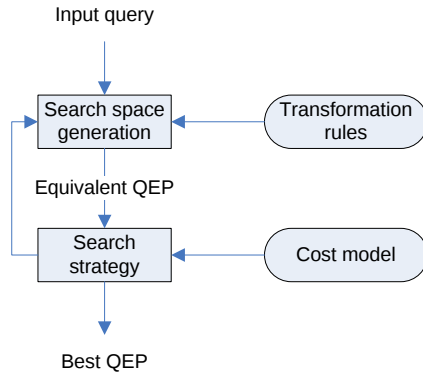


Fig. 1 Query optimization process

For a given query, the *search space* can be defined as the set of equivalent operator trees that can be produced using transformation rules. The example below illustrates 3 equivalent join trees, which are obtained by exploiting the associative property of binary operators. Join tree (c) which starts with a Cartesian product may have a much higher cost than other join trees.

```

SELECT  ENAME, RESP
FROM    EMP, ASG, PROJ
WHERE   EMP.ENO=ASG.ENO
AND     ASG.PNO=PROJ.PNO
  
```

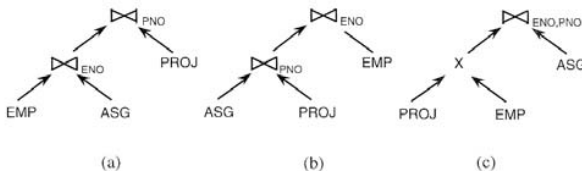


Fig. 2 Query equivalent trees

Regarding different search spaces, there would be different shape of the join tree. In a *linear tree*, at least one operand of each operand node is a base relation. However, a *bushy tree* might have operators whose both operands are intermediate operators. In a distributed environment, bushy trees are useful in exhibiting parallelism [4].

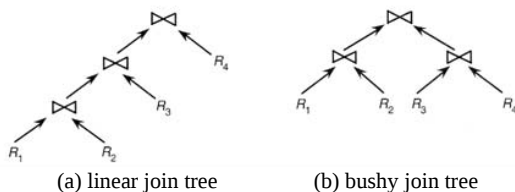


Fig. 3 Linear vs. bushy join tree

Considering new large scale database applications such as deductive database systems and bioinformatics, it is necessary to be able to deal with larger size queries. The search complexity constantly increases and makes higher demand for better algorithms than our traditional relational database queries.

III. RELATED WORKS

Three most common types of algorithms for join-ordering optimization are deterministic, Genetic and randomized algorithms [5].

Deterministic algorithm, also known as exhaustive search *dynamic programming* algorithm, produces optimal left-deep processing trees with the big disadvantage of having an exponential running time. This means that for queries with more than 10-15 joins, the running time and space complexity explodes [5]. Due to the very large time and space complexity of this algorithm for plan enumeration, iterative dynamic programming approach was proposed which produces reasonable plans with reasonable running time for most network topologies. However, its complexity is not much more than classical DP algorithm.

Genetic and randomized algorithms [6]-[7] on the other hand do not generally produce an optimal access plan. But in exchange they are superior to dynamic programming in terms of running time. Experiments have shown that it is possible to reach very similar results with both genetic and randomized algorithms depending on the chosen parameters. Still, the genetic algorithm has in some cases proved to be slightly superior to randomized algorithms.

Layers of distributed query optimization have been depicted in Fig. 4.

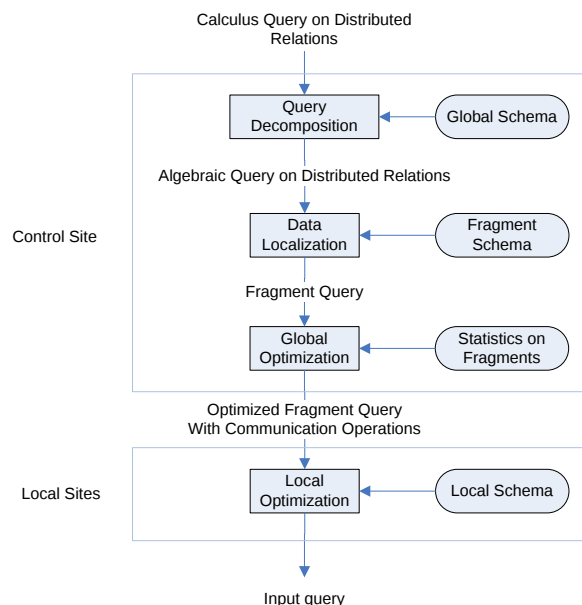


Fig. 4 Distributed query optimization

There are number of Query Execution Plan for DDB such as: row blocking, multi-cast optimization, multi-threaded execution, joins with horizontal partitioning, Semi Joins, and Top n queries [8]-[9]-[10]. In this paper we propose a novel agent based QEP generator for heterogeneous distributed data base systems.

Distributed Cost Model

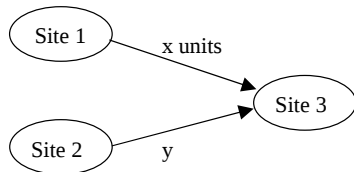
An optimizer cost model includes cost functions to predict the cost of operators, and formulas to evaluate the sizes of results. Cost functions can be expressed with respect to either the total time, or the response time [4]-[11]. The total time is the sum of all times and the response time is the elapsed time from the initiation to the completion of the query. The total time (TT) is computed as bellow, where T_{CPU} is the time of a CPU instruction, $T_{I/O}$ the time of a disk I/O, T_{MSG} the fixed time of initiating and receiving a message, and T_{TR} the time it takes to transmit a data unit from one site to another.

$$TT = T_{CPU} * \#_{insts} + T_{I/O} * \#_{I/Os} + T_{MSG} * \#_{msgs} + T_{TR} * \#_{bytes}$$

When the response time of the query is the objective function of the optimizer, parallel local processing and parallel communications must also be considered. This response time (RT) is calculated as bellow:

$$RT = T_{CPU} * seq_ \#_{insts} + T_{I/O} * seq_ \#_{I/Os} + T_{MSG} * seq_ \#_{msgs} + T_{TR} * seq_ \#_{bytes}$$

Most early distributed DBMSs designed for wide area networks have ignored the local processing cost and concentrate on minimizing the communication cost. Consider the following example:



$$TT = 2 * T_{MSG} + T_{TR} * (x + y)$$

$$RT = \max \{ T_{MSG} + T_{TR} * x, T_{MSG} + T_{TR} * y \}$$

In parallel transferring, response time is minimized by increasing the degree of parallel execution. This does not imply that the total time is also minimized. On contrary, it can increase the total time, for example by having more parallel local processing (often includes synchronization overhead) and transmissions. Minimizing the total time implies that the utilization of the resources improves, thus increasing the system throughput. In practice, a compromise between the total and response times is desired.

Database Statistics

The main factor affecting the performance is the size of the

intermediate relations that are produced during the execution. When a subsequent operation is located at a different site, the intermediate relation must be transmitted over the network. It is of prime interest to estimate the size of the intermediate results in order to minimize the size of data transfers. The estimation is based on statistical information about the base relations and formulas to predict the cardinalities of the results of the relational operations.

Cartesian product: The cardinality of the Cartesian product of R and S is $card(R \times S) = card(R) \times card(S)$

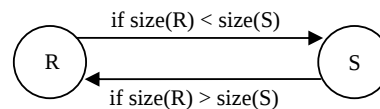
Join: There is no general way to estimate the cardinality of a join without additional information. The upper bound of the join cardinality is the cardinality of the Cartesian product. Some systems, such as Distributed INGRES [12], use this upper bound, which is quite pessimistic. R* [13] uses this upper bound divided by a constant to reflect the fact that the join result is smaller than the Cartesian product. However, there is a case, which occurs frequently, where the estimation is simple. If relation R is equi-join with S over attribute A from R, and B from S, where A is a key of relation R, and B is a foreign key of relation S, the cardinality of the result can be approximated as $card(R \bowtie_{A=B} S) = card(S)$

In other words, the Cartesian product $R \times S$ contains $n_r * n_s$ tuples; each tuple occupies $s_r + s_s$ bytes. If $R \cap S = \emptyset$, then $R \bowtie S$ is the same as $R \times S$. If $R \cap S$ is a key for R, then a tuple of s will join with at most one tuple from R, therefore, the number of tuples in $R \bowtie S$ is no greater than the number of tuples in S. If $R \cap S$ in S is a foreign key in S referencing R, then the number of tuples in $R \bowtie S$ is exactly the same as the number of tuples in s. The case for $R \cap S$ being a foreign key referencing S is symmetric.

As discussed earlier, ordering joins is an important aspect of centralized query optimization. This matter in a distributed context is even more important since joins between fragments may increase the communication time. Two main approaches exist to order joins in fragment queries:

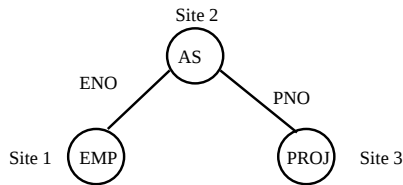
- 1) Direct optimization of the ordering of joins (e.g. in the Distributed INGRES algorithm).
- 2) Replacement of joins by combination of *semi-joins* in order to minimize communication costs.

Let R and S are relations stored at different sites and query $R \bowtie S$ be the join operator. The obvious choice is to send the smaller relation to the site of the larger one.



More interesting is the case where there are more than two relations to join. The objective of the join ordering algorithm is to transmit smaller operands. Since the join operations may reduce or increase the size of intermediate results, estimating the size of joint results is mandatory, but difficult. Consider

the following query expressed in relat. alg.: $PROJ \bowtie_{PNO} EMP \bowtie_{ENO} ASG$ whose join graph is below:



This query can be executed in at least 5 different ways.

1. $EMP \rightarrow ASG$; $EMP' = EMP \bowtie ASG \rightarrow PROJ$; $EMP' \bowtie PROJ$
2. $ASG \rightarrow EMP$; $EMP' = EMP \bowtie ASG \rightarrow PROJ$; $EMP' \bowtie PROJ$
3. $ASG \rightarrow PROJ$; $ASG' = ASG \bowtie PROJ \rightarrow EMP$; $ASG' \bowtie EMP$
4. $PROJ \rightarrow ASG$; $PROJ' = PROJ \bowtie ASG \rightarrow EMP$; $PROJ' \bowtie EMP$
5. $EMP \rightarrow ASG$; $PROJ \rightarrow ASG$; $EMP \bowtie PROJ \bowtie ASG$

To select one of these programs, the following sizes must be known or predicted: $size(EMP)$, $size(ASG)$, $size(PROJ)$, $size(EMP \bowtie ASG)$, $size(ASG \bowtie PROJ)$. Furthermore, if it is the response time that is being considered, the optimization must take into account the fact that transfers can be done in parallel with strategy 5. An alternative to enumerating all the solutions is to use heuristics that consider only the size of the operand relations by assuming, for example, that the cardinality of the resulting join is the product of cardinalities. In this case, relations are ordered by increasing sizes and the order of execution is given by this ordering and the join graph. For instance, the order $(EMP, ASG, PROJ)$ could use strategy 1, while the order $(PROJ, ASG, EMP)$ could use strategy 4.

Multi Agent Systems

Multi-agent systems (MASs) as an emerging sub-field of artificial intelligence concern with interaction of agents to solve a common problem [14]. This paradigm has become more and more important in many aspects of computer science by introducing the issues of distributed intelligence and interaction. They represent a new way of analyzing, designing, and implementing complex software systems.

In multi-agent systems, communication is the basis for interactions and social organizations which enables the agents to cooperate and coordinate their actions. A number of communication languages have been developed for inter-agent communication, in which the most widely used ones are KIF (Knowledge Interchange Format) [15], KQML (Knowledge Query and Manipulation Language) [16], and ACL (Agent Communication Language) [17]. KQML uses KIF to express the content of a message based on the first-order logic. KIF is a language intended primarily to express the content part of KQML messages. ACL is another communication standard emerging in competition with KQML since 1995. Nowadays, XML (Extensible Markup Language) started to show its performance as a language to encode the messages exchanged between the agents, in particular in agent-based e-commerce

IV. PROPOSED SYSTEM ARCHITECTURE

Although the problem of distributed query processing in heterogeneous systems has been investigated before [19], a good solution to practical query optimization has not been studies well. To meet so we proposed new multi-agent system architecture based on *Java Agent DEelopment* (JADE) framework [20]. JADE is a software development framework aimed at developing multi-agent systems and applications in which agents communicate using FIPA¹ Agent Communication Language (ACL) messages and live in containers which may be distributed to several different machines. The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform.

Each agent must register with an AMS in order to get a valid AID. The Directory Facilitator (DF) is the agent who provides the default yellow page service in the platform. The Message Transport System, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms.

JADE is capable of linking Web services and agents together to enable semantic web applications. A Web service can be published as a JADE agent service and an agent service can be symmetrically published as a Web service endpoint. Invoking a Web service is just like invoking a normal agent service. Web services' clients can also search for and invoke agent services hosted within JADE containers.

The Web Services Integration Gateway (WSIG) [21] uses a Gateway agent to control the gateway from within a JADE container. Interaction among agents on different platforms is achieved through the Agent Communication Channel. Whenever a JADE agent sends a message and the receiver lives on a different agent platform, a Message Transport Protocol (MTP) is used to implement lower level message delivery procedures [22]. Currently there are two main MTPs to support this inter-platform agent communication - CORBA IIOP-based and HTTP-based MTP.

Considering large-scale applications over separated networks, agent communications has to be handled behind firewalls and Network Address Translators (NATs), however, the current JADE MTP do not allow agent communication through firewalls and NATs. Fortunately, the firewall/NAT issue can be solved by using the current JXTA implementation for agent communication [23].

JXTA is a set of open protocols for P2P networking. These protocols enable developers to build and deploy P2P applications through a unified medium [24]. Obviously, JXTA is a suitable architecture for implementing MTP-s for JADE and consequently JADE agent communication within different networks can be facilitated by incorporating JXTA technology into JADE [23].

¹ Foundation for Intelligent Physical Agents (<http://www.fipa.org>)

In this work, a query is submitted by user to the Query Distributor Agent and then it will be distributed using the submitted terms in the available ontologies. Our proposed architecture uses different types of agents, each having its own characteristics. The proposed system architecture is shown in Fig. 5.

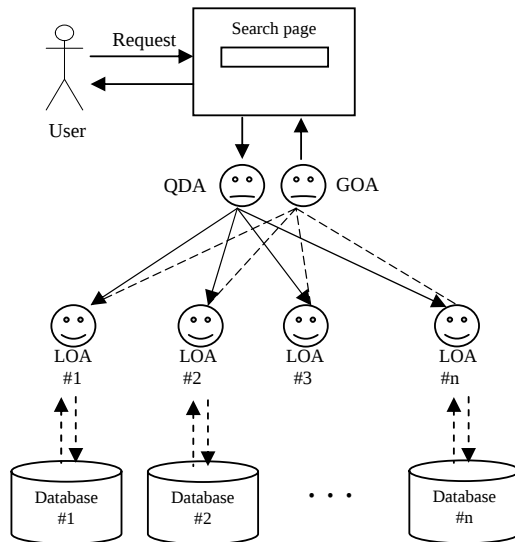


Fig. 5 Proposed system architecture

1. Query Distributor Agent (QDA): After receiving the user query, the QDA sends sub-queries to responsible local optimizer agents. The QDA can also create search agents if needed.

2. Local Optimizer Agents (LOAs): These agents apply a Genetic algorithm based sub-query optimization which will be discussed later, and return a result table size to the Global Optimizer Agent.

3. Global Optimizer Agent (GOA): This agent has the responsibility to find best join order via network. To do so, GOA receives result table size information from LOAs and again using an evolutionary method finds a semi-optimal join order, however, this time the GA fitness function is based on minimizing communication rate among different sites. The final result would be then send to LOAs to perform operation and deliver result to the GOA to show them on the screen.

Genetic Algorithm based Optimization

The basic concept of the GAs were originally developed by Holland [25] and later revised by Goldberg [26]. Goldberg showed that GAs are independent of any assumption about the search space and are based on the mechanism of natural genetics. The first step to model this problem as a GA problem, is determining the *chromosome*, *GA operators*, and *fitness function*.

a) Chromosome Design

In order to encode different access plans, we considered *ordered list scheme* to represent each processing tree as an ordered list. For instance $((((R1 \bowtie R5) \bowtie R3) \bowtie R2) \bowtie R4)$ is encoded as "15324". This converts the join order to the well-known traveling salesman problem (TSP). Possible query plans are encoded as integer strings. Each string represents the join order from one relation of the query to the next. This mechanism is also used within the PostgreSQL optimizer [27]. There may be other ways to encode processing trees especially bushy trees; however, we have used the above explained ordered list encoding method for implementation simplicity and faster run.

b) GA operations

For the crossover, one point in the selected chromosome would be selected along with a corresponding point in another chromosome and then the tails would be exchanged. Mutation processes causes some bits to invert and produces some new information. The only problem of mutation is that it may cause some useful information to be corrupted. Therefore we used elitism which means the best individual will go forward to the next generation without undergoing any change to keep the best information.

c) Fitness function

Defining fitness function is one of the most important steps in designing a GA-based method, which can guide the search toward the best solution. In our simulation we used a simple random cost estimator as defined below where $\text{random}(x)$ returns an integer between 0 and x .

$$\text{fitness} = \begin{cases} \text{random}(R \times S) + (R + S) & ; \text{if } (R + S) > (R \times S) \\ \text{random}(R + S) + (R \times S) & ; \text{else} \end{cases}$$

d) Algorithm design

After calculating the fitness function value for each parent chromosome the algorithm will generate N children. The lower a parent chromosome's fitness function value is the higher probability it has to contribute one or more offspring in the next generation. After performing operations, some chromosomes might not satisfy the fitness and as a result the algorithm discards this process and gets M ($M \leq N$) children chromosomes. The algorithm then selects N chromosomes with the lower fitness value from the $M + N$ chromosomes (M children and N parents) to be parent of the next generations. This process would repeat until a certain number of generations are processed, after which the best chromosome is chosen. Fig. 2 shows our GA based approach.

```

input: Relation size on different sites
output: Semi-Optimal join orders

initialize gene with a random join order
produce N initial parent chromosomes
mincost = d[gene.answer[0]]+d[gene.answer[1]];
maxcost = d[gene.answer[0]]*d[gene.answer[1]];
if (mincost>maxcost)
    gene.value = random(mincost-maxcost)+maxcost;
else
    gene.value = random(maxcost-mincost)+mincost;
while done ≠ yes do
    produce N random children chromosomes
    pass the best individual to next generation
    randomly mating
    exchange parts of chromosomes
    mutate with rate = 2/100
    for (i=1;i<n-1;i++)
    {
        mincost = prev_join+d[gene.answer[i+1]];
        maxcost = prev_join*d[gene.answer[i+1]];
        if (mincost>maxcost)
            gene.value = random(mincost-maxcost)
                        + maxcost;
        else
            gene.value = random(maxcost-mincost)
                        + mincost;
    }
    if gene.value satisfied then done = yes
    else produce next generation chromosomes
end while

```

Fig 2 Our GA-based query optimization algorithm

V. RESULTS

Experiments were done on a PC Pentium 4 CPU 2 GHz and 1GB RAM. The evolutionary process accomplished in less than a second and seen in Fig. 6 (a) sample query for 20 joins is converged to a near optimal fitness almost after 100 generations. Table I shows the parameters value we set for our implementation.

TABLE I
PARAMETERS SETTINGS FOR GA-BASED APPROACH

Parameter	Value
Population size	100
Mutation probability	0.02
Crossover probability	0.7
Elitism probability	0.5
Number of generations	100

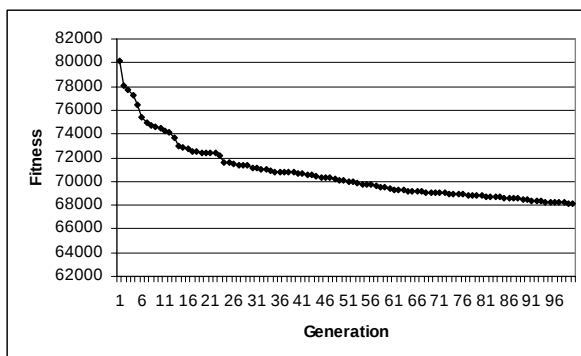


Fig. 6 Optimization of a 20-joins query in 100 generations

There is no doubt that dynamic programming method always gives us optimal solution, however, since the time and space complexity of the GA-base optimization is much less, it is not a practical approach for high amount of nested joins. Fig. 7 shows T_{GA} / T_{DP} which is rate of average run time for 10 queries, where T_{GA} is GA based optimization average run time and T_{DP} is dynamic programming based optimization average run time for the same query. Results as expected shows this rate is always less than 1 which means the GA approach has a less overhead in contrast with the DP method.

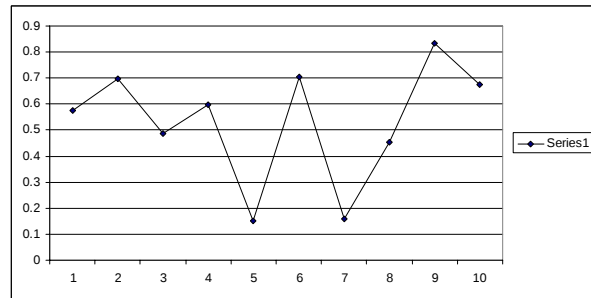


Fig. 7 Rate of optimization average run time for 10 queries

A computational complexity analysis in Table II shows the superiority of the evolutionary method in comparison with the dynamic programming approach. The time and space complexity in the evolutionary method is liner with respect to G as the number of generations, and N as the number of chromosomes in the population, while it is exponential in the DP method which is not efficient in handling more than 10 joins.

TABLE II
COMPUTATIONAL COMPLEXITY COMPARISON

	Space Complexity	Time Complexity
Evolutionary Method	$O(N)$	$O(G.N)$
Dynamic Programming	$O(2^n)$	$O(2^n)$

V. CONCLUSION

An evolutionary query optimization mechanism in distributed heterogeneous systems has been proposed using multi-agent architecture and genetic algorithm approach. Although deterministic dynamic programming algorithm produces optimal left-deep processing trees, it has the big disadvantage of having an exponential running time. Genetic and randomized algorithms on the other hand do not generally produce an optimal access plan. But in exchange they are superior to dynamic programming in terms of running time. Our practical framework uses hybrid JADE-JXTA framework which allows agent communication through firewalls and NATs in heterogeneous networks.

REFERENCES

- [1] J. Callan, "Distributed information retrieval". In *Advances in Information Retrieval*, W. B. Croft, Ed. Kluwer Academic Publishers, 2000, pp. 127–150.
- [2] L. Si, J. Callan, "A Semisupervised Learning Method to Merge Search Engine Results", *ACM Transactions on Information Systems*, Vol. 21, No. 4, October 2003, Pages 457–491.
- [3] Li, Victor O. K. "Query processing in distributed data bases", MIT. Lab. for Information and Decision Systems Series/Report no.: LIDS-P ; 1107, 1981
- [4] M. Tamer Özsu, Patrick Valduriez, "Principles of Distributed Database Systems, Second Edition", Prentice Hall, ISBN 0-13-659707-6, 1999
- [5] Kristina Zelenay, "Query Optimization", ETH Zürich, Seminar *Algorithmen für Datenbanksysteme*, June 2005
- [6] Yannis E. Ioannidis and Youngkyung Cha Kang, "Randomized Algorithms for Optimizing Large Join Queries"
- [7] Michael Steinbrunn, Guido Moerkotte, Alfons Kemper, "Heuristic and Randomized Optimization for the Join Ordering Problem", *The VLDB Journal - The International Journal on Very Large Data Bases*, Volume 6, Issue 3 (August 1997), Pages: 191-208, ISSN:1066-8888
- [8] D. Kossman, "The state of the art in distributed query processing" (*ACM Computing Surveys*, ISSN:0360-0300, 2000, Volume 32, Issue 4 December 2000, Pages: 422 - 469
- [9] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, "Access path selection in a relational database management system", *Morgan Kaufmann Series In Data Management Systems*, Readings in database systems (3rd ed.), Pages: 141 – 152, 1998, ISBN:1-55860-523-1
- [10] Stocker, Kossman, Braumandl, Kemper, "Integrating semi join reducers into state of the art query processors", (*ICDE 2001*)
- [11] Stefano Ceri, Giuseppe Pelagatti, "Distributed Databases: Principles and Systems", McGraw-Hill, ISBN-10: 0070108293, ISBN-13: 978-0070108295, 1984
- [12] The Design and Implementation of Distributed INGRES (context) - Stonebraker – 1986
- [13] Patricia G. Selinger, Michel E. Adiba, "Access Path Selection in Distributed Database Management Systems", *ICOD 1980*: 204-215
- [14] M. Wooldridge, "An Introduction to Multiagent Systems" Published in February 2002 by John Wiley & Sons. ISBN 0 47149691X.
- [15] M. R. Genesereth, R. E. Fikes, "Knowledge interchange format", version 3.0. Technical Report 92-1, Stanford University, Computer Science Department, 1992
- [16] T. Finin, R. Fritzson, D. McKay, R. McEntire, "KQML as an Agent Communication Language", *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, ACM Press, Gaithersburg, MD, USA, editor N. Adam, B. Bhargava, Y. Yesha, pp 456-463, 1994
- [17] Y. Labrou, T. Finin, Y. Peng, "The current landscape of Agent Communication Languages", *The current landscape of Agent Communication Languages*, *Intelligent Systems*, volume 14, number 2, March/April 1999, IEEE Computer Society, 1999
- [18] A. Korzyk, "Towards XML As A Secure Intelligent Agent Communication Language", the 23rd National Information Systems Security Conference, Baltimore Convention Center, Baltimore, Maryland, SA, October 16-19, 2000
- [19] Huang, Kuan-Tsae, Davenport, Wilbur B., "Query processing in distributed heterogeneous databases", MIT. Laboratory for Information and Decision Systems Series/Report no.: LIDS-P ;1212, 1981
- [20] F. Bellifemine, G. Caire, T. Trucco, G. Rimassa, "JADE Programmer's Guide", 21-August-2006.
- [21] JADE Board, "JADE WSIG Add-On Guide JADE Web Services Integration Gateway (WSIG) Guide", 03-March-2005
- [22] E. Cortese, F. Quarta, G. Vitaglione, P. Vrba. "Scalability and Performance of the JADE Message Transport System". *Analysis of Suitability for Holonic Manufacturing Systems*, exp. 2002.
- [23] S. Liu, P. Küngas, M. Matskin, "Agent-Based Web Service Composition with JADE and JXTA", *Proceedings of the 2006 International Conference on Semantic Web and Web Services, SWWS 2006*, Las Vegas, Nevada, USA, June 26-29, 2006
- [24] J. D. Gradecki, "Mastering JXTA: Building Java Peer-to-Peer Applications", JohnWiley&Sons, 2002.
- [25] J. H. Holland, "Adaptation in natural and artificial systems", *Ann Arbor, MI University of Michigan Press* 1975.
- [26] D. E. Goldberg, "The genetic algorithms in search, optimization, and machine learning", New York: Addison-Wesley, 1989.
- [27] PostgreSQL 8.3.0 Documentation, Chapter 49. Genetic Query Optimizer <http://www.postgresql.org/docs/8.3/interactive/geqo.html>