A New Allocation Technique for Methods and Attributes in Distributed Object-Oriented Databases Using Genetic Algorithms

Amany Sarhan
Faculty of Engineering,Tanta University, Egypt

Abstract: *With the wide increase in using distributed object-oriented databases, it became important to find an efficient technique to store large-scale databases on the different sites. The allocation of object-oriented database involves allocation of both methods and attributes within the classes. The main objective of this paper is to introduce a new technique for allocating the distributed object-oriented database methods and attributes among N sites. The proposed technique uses genetic algorithm to find the best allocation (optimal or near optimal) of the object-oriented database methods and attributes to the available sites. A cost function that computes the total data transfer during the execution of queries is developed. The genetic algorithm steps use this cost function to evaluate the possible allocations of methods. Validation of the proposed technique is done via simulation. The experimental results of the proposed technique depict that it has a great impact in reducing the total time required to find the best allocation and in most of the cases it reaches the optimal allocation of the methods.*

Keywords: *Object-oriented database, allocation, methods, attributes, genetic algorithm, distributed database.*

Received November 19, 2006; accepted June 17, 2007

1. Introduction

Database systems have been applied to a wide range of applications including, for example, computer-aided design and multimedia databases [8, 24]. Those new applications need to handle audio, video, text, and graphics data. The relational model due to its limitations is unable to manage such type of data. Thus, new data models, new query languages, new transaction models, and new fragmentation techniques are required for those applications. As a result, database researchers introduced new data models, such as the object-oriented, to overcome the relational model's limitations [3, 5, 6, 8, 11]. However, the object-oriented model presented new features that were not available at the relational model such as inheritance, encapsulation, object identity and complex objects, which will require different techniques for data management [1, 3].

Accompanied with the database evolution is the vast need for distributed system producing the Distributed Database Systems (DDBS). DDBS is a decentralized DB system spread over different sites which are connected by underlining communication networks. Each site contains a portion of the global DB system. Since data and applications are spread over different sites over the network, DDBS can provide higher reliability and availability. DDBS also promises to provide improved performance obtained by distributed processing. Local queries and transactions tend to be much faster since the local database is just a portion of the global database and is much smaller compared to a centralized one [24].

However, DDBS adds more complexity in terms of its design and implementation [8, 24]. The design of distributed database enhances the performance of the application in two ways [20]: by reducing the amount of irrelevant data accessed by the applications and by reducing the amount of data transferred in processing the applications. There are two main activities in distributed database design: partitioning and allocation, which refers to partitioning the data and allocating fragments to various sites, respectively. The result of an allocation process is an allocation schema. The research work in relational database has proved that the allocation process is an NP complete problem [2, 5]. The distribution of database to the available sites is the backbone of the DDBS success. There exist many distribution possibilities, one of them can be several order of magnitudes faster than another. It is therefore important to carefully use a technique that finds a good (i.e. faster) distribution of data, ideally the best one if possible. Techniques that may assist designers in determining the optimal fragmentation and allocation of the fragments are important, especially in the early stages of designing a reliable distributed database system, which in turn will affect the overall performance for the system later on [4].

another class (called superclass) and adds to it. Inheritance is the ability of a subclass to receive all data and operations coming from its superclass.

The internal structure of the object is hidden from the user using what so called encapsulation. Encapsulation means an object contains both programs and data and offers to the world an interface and an implementation part called methods. The interface part is the specification of the set of operations that can be performed on the object; the implementation part describes the implementation of each operation. In most OODB, even data specification is part of the interface. Objects can be simple or complex; complex objects are built from simpler ones by applying constructors to them. The object constructors must be orthogonal, this means any constructor should apply to any object. Users of the system may define additional operations on complex objects.

2.1. Previous Work

In OODB the main objective of vertical fragmentation is to break a class into a set of smaller classes (called fragments) to permit user applications to execute using one fragment located at local sites which means minimum user application execution time [8, 9, 24]. The horizontal fragmentation in OODB aims to break the instants (objects) of the class into fragments to reduce the query execution time by minimizing the number of irrelevant objects accessed and reducing the data transfer among sites [20, 23]. Horizontal fragmentation is subdivided into two steps: primary and derived fragmentation. Primary fragmentation basically optimizes set operations over a class extension first by reducing the amount of irrelevant data accessed and secondly by permitting applications to be executed concurrently. The derived fragmentation clusters objects of distinct classes in the disk. The hybrid fragmentation in OODB divides the classes both vertically then horizontally or vice versa. Vertical class partitioning is a type of vertical fragmentation that partitions the database classes to reduce irrelevant disk-stored data in local site [7, 14, 15, 18, 19].

There is only little research work directed towards the allocation of OODB methods and attributes relatively to those directed to the relational model due to the complexities added by the object-oriented model. Some of the previous work like [11, 12, 23] concentrated on allocating fragments produced by other fragmentation techniques. The fragmentation techniques are first used to obtain methods and attributes fragments as in [11, 12]. Then an allocation technique is used to allocate those fragments. The research work in [15] tackled the problem of vertical class partitioning and based the partitioning algorithm on a developed cost model rather than affinity.

However, little work focused on the method allocation problem for OODB such as [4, 5, 20]. Karlapalem *et al.* [2] tackled the problem of the allocation in two directions: class fragments allocation and methods allocation. However, their algorithm was not effective to solve the problem. In [5] the problem of class fragments

Traditionally in relational database, fragmentation can be classified into three types: vertical, horizontal, and hybrid. The vertical fragmentation aims to break up a relation into a set of relations while the horizontal fragmentation aims to break the large number of instants into disjoint subsets each of which will be allocated to different sites [1, 21, 22, 24]. The hybrid fragmentation first divides the relation horizontally, and then splits each of the obtained fragments vertically or vice versa. Many allocation techniques have been introduced for allocating the produced fragments [2, 10, 21]. In [2] the allocation technique aimed to minimize the total transfer cost, while [10] proposed a strategy that integrates the treatment of relation assignment and query strategy to optimize the performance of DDBS. *A* site-independent fragment dependency graph representation was developed in [21] to model the dependencies between the fragments. Then the data allocation problem was formulated as a mapping problem, which can be mapped to a maximum flow minimum cost problem to achieve an optimal solution.

In Object-Oriented DataBases (OODB), many research have been introduced in the field of vertical, horizontal, mixed fragmentation [7, 11, 12, 14, 20, 23]. Others have concentrated on the allocation of these fragments [8, 9] while few concentrated on the direct allocation of methods and attributes [4, 5, 20].

In this paper, a new technique is proposed for the allocation of OODB methods and attributes to a number of given sites based on Genetic Algorithm (GA). The main objective of this technique is to obtain an allocation schema that minimizes the total data transfer cost. The proposed technique, while using GA for search [9, 13, 17], aims to overcome the problems associated with previous search techniques, such as: hill-climbing [4], exhaustive search or Branch and Bound. Through iterative steps, the genetic algorithm searches the solution space for possible allocations, evaluating them using a developed cost function and finally finds the best allocation schema that in most cases is the optimal one.

2. Basic Object-Oriented Databases Concepts

The OODB differs from the relational databases in that instead of having records, OODB has objects. Each object has an object identity. Identity is that property of an object, which distinguishes it from all other objects [3, 6, 16]. Any object belongs to a specific class. *A* class is a set of objects that share a common structure and a common behavior. *A* class summarizes the common features of a set of objects; a class has attributes, which constitutes the data within the object, and a set of operations using which the user can manipulate the object's attributes. A class (called subclass) can inherit some (or all) of the properties of

the best allocation schema in smaller time and with the same accuracy of the optimal one. This is important especially for the situations of rapid changing database systems.

3.1. The Proposed Allocation Technique

This paper presents a new technique for allocation of object-oriented database methods and attributes to a number of given sites based on GA. The proposed technique while using GA for search aims to overcome the problems associated with previous techniques, such as: exhaustive search, Branch and Bound and hill-climbing. Through iterative steps, the genetic algorithm searches the solution space for possible allocations, evaluating them using a developed cost function and finally finds the best allocation that in most cases is the optimal one. The GA uses a cost function that is developed through this paper to evaluate the quality of each possible allocation in terms of total methods execution time. This cost function computes the total cost of executing the queries in terms of data transferred between the sites participating in executing the queries.

The proposed technique uses two different algorithms for allocation. First, it uses GA to obtain an allocation of database methods to the available sites. For each possible methods allocation, the attributes of the database are allocated using method-attribute affinity approach found in [11, 12]. The GA uses a cost function to evaluate the possible allocations. Through the iterative steps of GA, the search space of the problem is explored and a final best solution that may be the optimal or near optimal one is provided.

3.2. Developing the Cost Function

The most significant cost to consider in DOODB system is the cost of transferring data between the sites. The derived cost function in this paper computes the total methods' cost incurred by all methods in the database when it is called from the different sites. The cost of calling a method is simply expressed by the sum of all communicational cost (data transferred) between the site containing this method and the sites calling it. Thus, when computing the execution time of any method, its type should be identified first. If it is a simple method, then the only cost to consider is the cost of transferring the resultant data from executing the method to the calling site. While if it is a complex one, the cost of the invoked methods should be included in the cost equation. We also have to know if it requires any remote objects transfer. This occurs when the method requires objects for execution that is not available on this site. Thus, it has to import it from another site to complete the execution of the method. If so, the cost of transferring these objects is also included in the cost function. Therefore, the cost of any method may include the following terms [4]:

- Cost of data returned to the calling sites

allocation in Distributed Object Oriented Database (DOODB) has been addressed. They introduced a general taxonomy based on the data model, degree of redundancy and design objectives. A valuable work introduced by Bellartreche *et el*. [4] considered the allocation of OODB methods and attributes using hill-climbing algorithm that enhances the quality of an initial allocation solution. They proved that through hill climbing technique, the initial allocation solution can be improved and in many cases reach the optimal solution. They developed and used a cost function to evaluate the solutions. However, hill-climbing algorithm has the disadvantage of getting stuck in local minima while the search for optimal solution. Another possible way of finding the optimal allocation is to use either exhaustive search or Branch and Bound search technique. Both of them suffer from long search time.

2.2. Method Execution in DOODS

In object-oriented classes, there are basically two kinds of methods; simple and complex. Simple methods are those that do not invoke other methods of other classes when executing. Complex methods are those that can invoke methods of the same or of other classes when executing. Those invoked methods may also be simple or complex and so on. User applications that access attributes and methods of a class are of three types namely as follows [11, 12]:

- Those running directly on this class.
- Those running on descendants of this class.
- Those running on methods of other classes in the database that use methods of this class.

Another issue to consider is that during the execution of any method there could be a need for objects that are not available on the site containing this method (remote objects). Thus, these objects must be transferred to the executing site in order to complete the method's execution.

3. Problem Definition

In this paper, the problem of distributing (allocation) the OODB methods and attributes to the available sites was tackled. The distribution process aimed to find the best (possibly the optimal) allocation of methods and attributes that gives the minimum query execution time for a specific set of queries. One possibility is to use a traditional search technique that will enumerate all the possible allocation solutions that are then evaluated to get the best one. Unfortunately, for the current available OODBs that contain large number of methods and attributes, this process can take considerable execution time. Thus, heuristic algorithms, such hill-climbing, were proposed to solve this problem. However, these techniques lack the accuracy where in many cases they do not reach the optimal solutions. So, a fast yet efficient search technique is required that enables the designer to find

m_k allocated to site S_i, as illustrated in Figure 1, can be given by the following equation:

$$\mathrm{COST}(\mathrm{m_k}) = \sum_{j=1}^{N} \sum_{l=1, l\neq i}^{S} \frac{MU(q_j, m_k) * AF(q_j, s_l) * }{SDT(s_i, s_l)} \quad (1)$$

3.4. Cost Function for Remote Objects Transfer

Symbol	Description
q_j	Query number j
N	Total number of queries
m_i	Method number i
M	Total number of methods in the database
S_i	Site number i
S	Total number of sites
AF(q_i ,S_j)	The applications frequency of users queries (q_i) from different sites S_j
SDT(S_i ,S_j)	The speed of data transferred between sites (S_i, S_j) per unit time where S_i is client site and S_j is server site
MU(q_i,m_j)	The method usage of query q_i to method m_j. MU(q_i,m_j)= 1 when query q_i invokes method m_j, otherwise MU(q_i,m_j)=0.
ADT(m_j)	The amount of data returned after the execution of method m_j.
MMR(m_i, m_j)	Method-Method Reference, MMR(m_i, m_j)=1 if method m_i is a complex method that invokes method m_j. If method m_i is simple, this value =0.
Obj_size(m_j)	Size of objects required to execute method m_j
MAR(M_j , A_i)	Method-Attribute reference that determines the attributes referenced (used) by a method use.

The cost of transferring the remote objects to the executing site depends on the size of objects transferred and the speed of data transfer between the two sites. It also depends on the access frequency of the applications to this method and the method usage by the application. We have to note that there could be more than one source of remote objects, thus the equation will sum all costs of remote objects transfer as follows (Figure 2):

$$\mathrm{COST_o}(\mathrm{m_k}) = \sum_{j=1}^{N} \sum_{l=1, l\neq i}^{S} \frac{MU(q_j, m_k) * AF(q_j, s_l) * Ob}{SDT(s_i, s_l)} \quad (2)$$

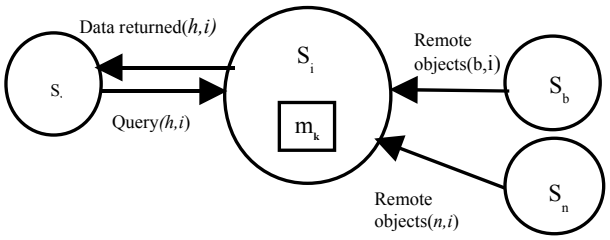


Figure 2. Remote objects transferred between Si and other sites.

- Cost of remote objects transfer
- Cost of invoked methods

Each of these costs will be separately considered in the following sections. For simplicity, all the methods in the database are grouped and given a unique single subscript (m_i) instead of two subscript ($m_{i,j}$). In order to compute this cost, we first have to define the parameters used as illustrated in Table 1.

Table 1. Summary and description of parameters used.

3.3. Cost Function for Simple Method

For a simple method, the only type of cost accounted for is the cost of returning the final results to the calling site. To clearly understand the cost incurred during calling a method from other sites, let's have an example as shown in Figure 1. If method m_k is allocated to site S_i, the cost of executing this method will be the sum of all the costs of transferring the data returned to the calling sites from the site containing the method (sum of data returned(*h, i*) and data returned(*b, i)).*

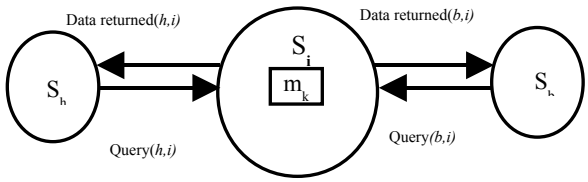


Figure 1. Query and data transferred between *Si* and other sites.

The cost of returning the resultant data to the calling site is based on the following criteria:

- Method Usage (MU) by the application which represents if or not this method is used by a query.
- Access Frequency (AF) of the applications which represents the number of times each query is initiated.
- Amount of Data Returned (ADT) after the execution of this method.
- Speed of Data Transferred (SDT) between the pairs of sites which sometimes called the link capacity.

So, for any query in the system, if we can determine whether or not it uses the method m_k (where its MU=1), and for this method if we know the amount of data returned after executing it (ADT), we can compute the communication cost of executing this method by dividing (ADT) by the speed of data transferred (SDT) between these two site. Multiplying this cost by the access frequency of the query from this site, we can compute the total cost due to this query. If we added all costs values for all queries from all sites that use m_k, we can get the total cost of this method. Thus, the cost of executing a simple method

also may need remote objects from other sites which will require adding the cost of remote objects transfer to its cost equation. Thus, in general, we can express the cost of invoking a method as:

$$COST_d(m_k) = COST_d(m_k) + COST_o(m_d) + COST_d(m_d) \quad (4)$$

where the first term, which accounts for the basic cost of the invoked method(s) required by m_k, is computed using equation 3, the second term, which accounts for the possible remote objects cost by the invoked method m_d, is computed using equation 2 and the third term, which accounts for any invoked method(s) called by *md* itself, is computed using equation 3 in recursive manner.
Finally, the total cost of any method is expressed as:

$$COST(m_k) = COST_s(m_k) + COST_o(m_k) + COST_d(m_k) \quad (5)$$

The last two terms may be zero if the method is simple. The cost function that computes the total cost of all methods in the database (simple or complex) is given by:

$$\text{TOTAL\_COST} = \sum_{k=1}^{m} COST(m_k) \quad (6)$$

Equation 6 represents the general form of total methods cost that is used by the GA to evaluate the possible allocations.

3.6. Method-Attribute Affinity Approach

After generating the methods allocation, the second stage of the proposed technique extends each method allocation produced in the previous stage to incorporate all attributes accessed by methods on this site. This is accomplished based on the affinity between the attributes and the methods. Also for simplicity, all the attributes in the database are grouped and given a unique single subscript (A_i) instead of two subscript ($A_{i,j}$). The following steps are used to generate the attributes allocation:

- The method-attribute reference information of the methods is used to include for each site all attributes accessed by methods on this site.
- There are cases when some attributes overlap in more than one site that occurs when the same attribute of a class belongs to the method attribute reference sets of two different methods at two different sites. Since our objective is to find a final non-overlapping method/attribute allocation, a technique is needed to decide at which site it is most beneficial to keep an overlapping attribute.

To solve such problem we use the Affinity Rule in [11] to decide which site will keep each overlapping attribute. This rule determines the affinity between the overlapping attribute and each of the sites containing it using the Attribute Fragment Affinity (AFA) statistics.

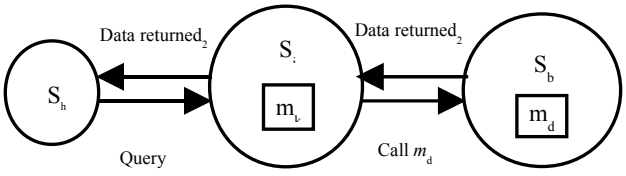
3.5. Cost Function for Invoked Methods

For complex methods, the cost equation becomes very complicated. A complex method calls (invokes) other methods (called invoked methods) during its execution. This type of relation between methods is called method-method dependency where a complex method mi calls another method *md* in the same class or in a different class. The dependency of methods can be detected through the Method-Method Reference (MMR) matrix as described in Table 1. We have to include the effect of a complex method by adding the cost of its invoked methods to the cost equation.

To better understand the terms participating in computing the complex method's cost, let's assume that m_k is a complex method allocated to site S_i. This method uses method m_d allocated at site S_b during its execution. Figure 3 illustrates the scenario of querying m_k from a third site S_h. There are two amount of data transferred during the execution of the complex method m_k: data returned$_1$ and data returned$_2$. The first amount (data returned$_1$) is returned from site S_b (that holds the invoked method m_d) to S_i (that holds the complex method m_k) after executing m_d. This partial results is required to complete the execution of the complex method m_k. While the second amount (data returned$_2$) is the final results returned to site S_h from S_i after executing m_k.

Thus, the cost of executing m_d is the cost of returning the partial results (data returned$_2$) to the site holding the complex method. This cost depends on the amount of data returned and the speed of data transfer between the two sites. Note that this cost will equal to zero if both m_k and m_j are allocated to the same site and if method m_k is simple method (as MMR=0 means no invoked methods). If the complex method invokes more than one method then the cost will e the sum of all these costs as follows:

$$COST_d(m_k) = \sum_{j=1}^{N} \sum_{d=1, i \neq b}^{M} \frac{AF(q_j, s_1) * ADT(m_d) * MMI}{SDT(s_i, s_b)} \quad (3)$$



mk : complex method
md: invoked method
Sh : querying site
Si: allocation site of mk
Sb: allocation site of mj

Figure 3. Data transferred when executing a complex method.

In many cases the invoked method itself is a complex method, thus its cost value must include the cost of the invoked methods it requires and so on. It

4. Repeat steps (2, 3) until number of iterations is finished.
5. Display the best answer found (which has the highest-fitness).

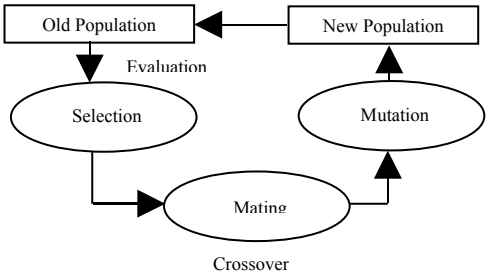


Figure 4. The basic genetic steps.

There are two key issues to be considered before using GA, which are how to represent the solution in GA domain and how to evaluate each solution.

4.1. Solution Representation in GA Domain

GA works on a population set of strings (or individuals). Each string of a population is a proposed solution and is encoded based on input data of the target application. In our work, a string is divided into *M* elements (parts) each corresponds to a method in the database. Each element will hold an integer number that represents the site number at which this method is proposed to be allocated. Thus, if we have a total of *M* methods in the database to be allocated to *S* available sites, the solution string is *M* bits long and the numbers found in each element will be between *1* and *S*. Figure 5 shows the solution representation of the problem.

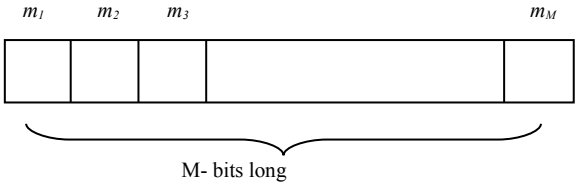


Figure 5. Solution representation in GA domain.

As an example, let's assume having 5 methods and 3 sites. One of the proposed allocation solutions is shown in Figure 6. This solution indicates that *m1* and *m4* are allocated to *site 1*, *m2* and *m5* are allocated to *site 2*, and *m3* is allocated to site 2 (as shown in Figure 7).

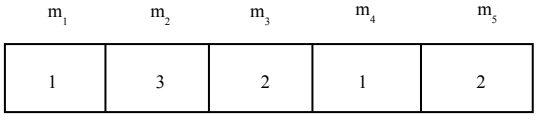
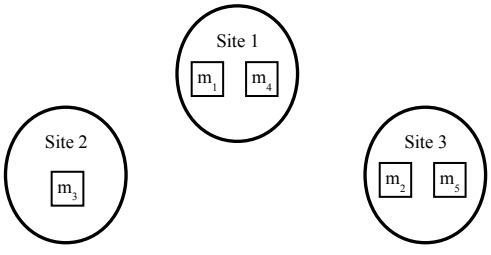


Figure 6. An example of solution representation.



Attribute Fragment Affinity AFA(*Ai,Sj*) is a measure of the affinity between an attribute Ai and the methods allocated on a specific site *Sj*. For each attribute, the affinity is measured to all the available sites using the following equation [11, 12]:

$$\mathrm{AFA}(\mathrm{A_i},\mathrm{S_j}) = \sum_{k=1}^{M}\sum_{p=1}^{N} \sum_{MAR(A_i,M_K)\wedge MU(q_p,M_k)=1} \sum_{l=1}^{S} \mathrm{AF}(\mathrm{q_p},\mathrm{S}$$

(7)

where *m* is the number of methods on the site S_j, *N* is the number of user queries and S is the number of sites, $MAR(A_i, m_k)$ is the attributes accessed by method m_k, $MU(q_p, m_k)$ is the method usage of query q_p to method m_k, and $AF(q_p, S_l)$ is the application access frequency from site S_l. It then places the attribute on the site with the highest affinity measure produced by the previous equation and removes it from every other site.

4. Genetic Algorithms

Genetic Algorithms (*GAs*) are robust and adaptive methods for solving search and optimization problems. They navigate through the large search space of complex problems to get the best solution [9, 13, 17]. They overcome some of the problems of traditional search methods such as hill-climbing [4] which can cause missing the global optimum (best solution) by getting stuck at local optimum points. GA attempts to solve the problems in a fashion similar to the way in which human genetic process seems to operate. GA traditionally works with a population of items (individuals), as candidate solutions of search space. Population most commonly contains 10-200 items that are usually fixed-length strings where each string is an encoding of the problem input data. Fitness function is defined to evaluate the quality of the strings in population. For each string, the objective function is calculated which determines the fitness or quality value of that individual. Probabilistic rules (not deterministic) are used to direct the search for the best solution in the algorithm. The genetic algorithm uses the current population of individuals to create a new one such that the individuals in the new population are, on average, better than those in the current population.

GA has three operators in order to get the new population from the old one: selection and reproduction, crossover and mutation. These operators are used iteratively for a number of iterations called generations to reach the best solution as shown in Figure 4. Theoretically, the best number of generations is the one that makes 95% of the individual optimal solution. GA uses the described operators iteratively as follows:

1. Generate an initial population, repeating random strings of fixed size.
2. Do the selection, reproduction, crossover and mutation operations of the all population.
3. Replace the old population with the new one.

The selection and reproduction operator selects individuals from the old population according to their fitness to perform the crossover on them. The individuals with lower-fitness (lower cost value) will have a higher probability of being chosen than the higher-fitness (higher cost value) individuals. The lower-fitness individuals in our case represent the solutions with lower communication cost that we desire to keep.

4.3.3. The Crossover Operator

From those selected individuals, pairs are selected randomly to perform the crossover on them to reproduce a new pair of individuals. The crossover operator combines the different parts of the selected strings around the crossover point to form new pair of strings that it is assumed to have better fitness than their parents. Crossover can be one-point or two points. in the proposed algorithm, only one-point crossover is used as Fig. 8 shows. The crossover point is located after a number of elements in the solution string. The result of the mating (or crossover) is only accepted and added to the new population if it has better fitness than its parents. The new generated individuals will replace the poor performance individuals presented in the old population.

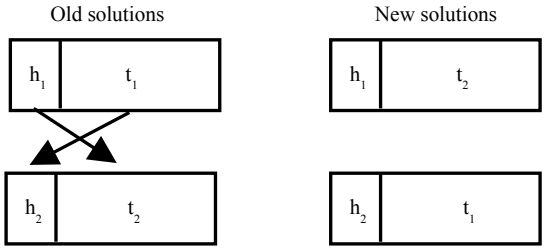


Figure 8. One-point crossover in GA.

4.3.4. The Mutation Operator

After performing the crossover and storing the new population, the mutation operator should be applied to the whole generated population. In our work, this step is only performed on the worst fitness individuals (having higher fitness which means higher total cost values) hoping to generate better individuals from them. At the same time, it provides a way to explore new regions of the search space and generate new alternatives without destroying the current good low-fitness solutions. One type of mutation operator (as Figure 9 shows) is to exchange the positions of two elements of the solution string. The two elements will be selected randomly from the string bits. The generated individual's fitness is computed and the individual is kept in its place in the population if it has better fitness than the old one. Otherwise, the old non-mutated individual is kept.

Performing all the operators on the old population will generate a new population. The new population will consist of the best individuals in the old population and the new generated individuals that has better fitness than the old ones. These individuals will replace the worst fitness individuals of the current

Figure 7. An allocation solution corresponding to the example.

4.2. Fitness Function Computation

For each solution (string), an objective function is calculated which determines the fitness or quality value of the solution. The objective function in our problem will be the total communication cost of that solution. The cost function developed in section 3.3 is used to evaluate the fitness of each of the proposed solution. The solution that has the minimum fitness will be the best solution.

4.3. The Genetic Algorithm Operators

The first basic step of the genetic algorithm is to initialize the population either totally random or partially random seeded with one or more initial solutions. The initial solution is obtained using an algorithm described later. The proposed allocation provides methods allocation. To complete the distribution of the database, the attributes have to be allocated. The allocation of attributes is performed using method-attribute affinity algorithm discussed earlier. The fitness function is then computed for each individual. A new population will be generated from the old one by applying the genetic algorithm operators to the old population. GA has three operators: selection and reproduction, crossover and mutation. These operators will be discussed next.

4.3.1. Generating an Initial Solution

Through previous use of GA in other research areas, I found that the convergence of GA towards the optimal solution is accelerated if it is seeded with at least one proper initial solution [13]. The algorithm used to generate the initial solution differs according to the application. In this work, the following steps are used to generate an initial allocation solution that will be added to the random solutions:

1. Consider all the methods in the database to be simple (for simplicity and speed).
2. For each method, compute the cost of allocation to each of the available sites using equation 1.
3. From these results, for each method, choose the site that gives the minimum cost for allocating this method.
4. Repeat steps 2 and 3 for all methods until each method has a specific site for allocation.
5. Represent the solution string as described in section 4.1

4.3.2. The Selection and Reproduction Operator

No. of Methods	No. of Sites	No. of Solutions	GA Time	B&B Time
5	3	15	120	530
10	4	37	230	4320
15	5	48	310	7250
20	6	69	370	9130
25	10	94	530	12450

From these results, it can be concluded that the proposed algorithm that uses GA effectively produced optimal results in most cases and near optimal results in few cases. In these cases, the algorithm can be repeated to get the optimal solution. However, the time required by the proposed algorithm is far smaller than the time required by Branch and Bound technique to produce the results.

Table 3. GA *vs* hill climbing produced solutions.

No. of Methods	No. of Sites	GA Time to Optimal	Hill Climbing Time to Optimal
5	3	120	210
10	4	230	290
15	5	310	Not reached
20	6	370	Not reached
25	10	530	Not reached

The proposed algorithm is also compared against the hill-climbing algorithm in [4] and the results are shown in Table 3. From these results we can see that GA outperforms hill-climbing algorithm, which in many cases fails to reach the optimal solution, in terms of the ability to reach the optimal solution and in term of time required to reach the optimal solution.

6. Conclusions

With the wide spread of databases, new generations of database model have evolved introducing the object-oriented database model. The object model is suitable for the new trend in applications that uses multimedia data instead of regular types of data. A problem associated with the massive amount of data is their distribution among different sites. Efficient allocation of data can drastically minimize the time required to execute any query by locating the right data in the right site and excluding the irrelevant data.

This paper introduced a novel technique for allocation of the object-oriented database methods that aimed to obtain the best allocation of methods and attributes. The optimal allocation is the one that has minimum data transfer cost among different sites thus minimum the query execution time. The proposed technique used the genetic algorithm to find the best allocation of methods and method-attribute affinity approach to find the attribute allocation. A cost equation is derived in this paper and used by the genetic algorithm search technique to evaluate the cost of the solutions. The experimental results verified that the proposed algorithm is capable of finding the best generation keeping the total number of individuals in each generation constant. The GA steps will be repeated until convergence to the optimal solution occurs or until the number of possible generations is reached. If convergence occurs, all the solutions in the new generation will be copies of one solution that is the optimal one.

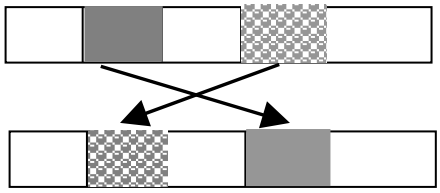


Figure 9. Mutation in GA.

In general, the larger the population size and the number of iterations used, the better the chance that an optimal solution will be found. A trade-off must be made between these two parameters (the population size and the number of iterations) and the execution time cost of the GA. The number of generations must be adjustable, too big numbers can take a long time to converge, while small one can cause missing the optimal solution by not converging. In our work, due to the precedence and the system constraints, the algorithm gives excellent results (as discussed next) in a reasonable search time.

5. Experimental Results and Analysis

The proposed GA was experimented using a number of randomly generated test cases. The data for the computation process for each test case such as number of classes, number of methods, number of sites, network link speeds, number of queries, etc …..are also randomly produced. The proposed technique starts by producing an initial solution, which is a possible allocation of methods to the available sites, after encoding the solution in GA domain. Then, GA's population is generated randomly and the initial solution is injected among the individuals. The algorithm when a convergence occurs or when the number of generations ends reaches a final solution. This solution represents the best allocation of the methods. However, the algorithm gives a list of obtained solutions produced through the search for further analysis by the designer. The produced solution is compared to the optimal solution produced by Branch and Bound technique.

Table 2 lists the results of the experiments conducted and shows the number of methods, number of sites, solutions generated by the proposed algorithm, the approximated time required to reach convergence to the optimal solution by GA algorithm and the time required to reach the optimal solutions by Branch and Bound.

Table 2. Results of the proposed technique and the Branch and Bound technique.

allocation of the methods to the available sites. The performance of the proposed algorithm also was compared against an existing algorithm that uses hill-climbing search technique and Branch and Bound. The results of the proposed algorithm showed that GA outperforms the technique that uses hill climbing for search in reaching better results for allocation in terms of required search time and the quality of the produced solution as it was able to reach the optimal solution without falling into a local minimum solution as hill-climbing did. It also showed better performance in terms of time required to reach optimal solution when compared to the Branch and Bound search technique.

References

[1] Agrawal S., Narasayya V., and Yang B., "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," *in Proceedings of the SIGMOD International Conference,* Paris, pp. 359-365, 2004.

[2] Apres P., "Data Allocation in Distributed Database Systems," *ACM Transactions on Database Systems*, vol. 13, no. 3, pp 263-304, 1988.

[3] Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D., and Zdonik S., *The Object-Oriented Database System Manifesto, Deductive and Object-Oriented Databases*, Elsevier Science Publishers, USA, 1989.

[4] Bellatreche L., Karlapalem K., and Li Q., "Complex Methods and Class Allocation in Distributed Object Oriented Databases," *in Proceedings of the 5th International Conference on Object Oriented Information Systems (OOIS'98)*, Paris, pp. 239-256, 1998.

[5] Bhar S. and Barker K., "Static Allocation in Distributed Object base Systems: A Graphical Approach," *in Proceedings of the 6th International Conference on Information System and Data Management, CISMOD'95, Lecture Notes in Computer Science 1006,* India, pp. 92-114, 1995.

[6] Burleson K., *Inside the Database Object Model*, CRC Press LLC, 1999.

[7] Chinchwadkar G. and Goh A., "An Overview of Vertical Partitioning in Object Oriented Databases," *Computer Journal*, vol. 42, no. 1, pp. 230-236, 1999.

[8] Connolly T. and Begg C., *Database Systems*, Pearson Education limited, 2004.

[9] Corcoran A. and Hale J., "A Genetic Algorithm for Fragment Allocation in a Distributed Database System," *in Proceedings of ACM Symposium on Applied Computing*, USA, 1994.

[10] Cornell D. and Yu P., "An Optimal Site Assignment for Relations in the Distributed Database Environment," *IEEE Transactions on Computers*, vol. 15, no. 5, pp. 1004-1009, 1989.

[11] Ezeife C. and Barker K., "Vertical Fragmentation for Advanced Object Models in a Distributed Object Based System," *in Proceedings of the 8th International Conference on Computing and Information*, pp. 50-67, Canada, 1995.

[12] Ezeife C. and Barker K., "Distributed Object Based Design: Vertical Fragmentation of Classes," *International Journal on Distributed and Parallel Databases*, vol. 6, no. 4, PP. 317-350, 1998.

[13] Fergany T. and Sarhan A., "Efficient Allocation of Distributed Object-Oriented Tasks to a Pre-Defined Scheduled System," *International Journal of Computers and Applications*, vol. 28, no. 1, pp. 35-42, 2006.

[14] Fung C., Karlapalem K., and Li Q., "An Evaluation of Vertical Class Partitioning for Query Processing in Object-Oriented Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1095-1118, 2002.

[15] Fung C., Karlapalem K., and Li Q., "Cost-Driven Vertical Class Partitioning for Methods in Object Oriented Databases," *VLDB Journal*, vol. 12, no. 3, pp.187-210, April 2003.

[16] Hull R., Tanaka K., and Yoshikawa M., "Behavior Analysis of Object-Oriented Databases: Method Structure, Execution Trees, and Reachability," *in Proceedings of the 3rd International Conference (FODO)*, pp. 39-48, New York, 1989.

[17] Jozef K., Dusan T., Vladimir F., and Ivana L., *Soft Computing in Industry: Recent Applications*, Springer, USA, 2002.

[18] Kamalakar K., Li Q., and Vieweg S., "Method Induced Partitioning Schemes in Object Oriented Databases," *in Proceedings of the 16th International Conference On Distributed Computing Systems,* pp. 377-384, Urbana Champaign, 1996.

[19] Karlapalem K. and Li Q., "A Framework for Class Partitioning in Object Oriented Databases," *Distributed and Parallel Databases*, vol. 8, no. 3, pp. 317-350, 1999.

[20] Karlapalem K., Navathe S., and Morsi M., *Distributed Object Management*, Morgan Kaufman Publishers Inc, USA, 1994.

[21] Navathe S. and Ra M., "Vertical Partitioning for Database Design: A Graphical Algorithm," *in Proceedings of ACM SIGMOD International Conference on Management of Data*, USA, pp. 544-546, 1989.

[22] Navathe S., Ceri S., Wiederhold G., and Dou J., "Vertical Partitioning Algorithms for Database Design," *ACM Transaction on Database Systems*, vol. 9, no. 4, pp. 680-710, 1984.

[23] Mattoso M., Baião F., and Zavrucha G., "Horizontal Fragmentation in Object DBMS: New Issues and Performance Evaluation," *in Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference*, California, 2000.

[24] Ozsu M. and Valduriez P., *Principles of Distributed Database Systems*, Prentice Hall, Englewood Cliffs, USA, 1999.

Amany Sarhan received the BSc degree in electronics engineering and MSc in computer science and automatic control from the Faculty of Engineering, Mansoura University, in 1990, and 1997, respectively. She awarded the PhD degree as a joint research between Tanta University, and Mansoura University Egypt.