

# Efficient Allocation in Distributed Object Oriented Databases

Jonathan Graham  
Center for Secure and Dependable Systems  
University of Idaho  
Moscow, ID, 83844 USA

## Abstract

Efficient distribution of data is a major challenge in distributed databases. The problem is even more severe for distributed object oriented databases because of inheritance, encapsulation and the more complex problem involved when methods invoke other methods. This problem is a harder version of the relational database allocation problem (DAP), a problem known to be NP-hard.

We looked at the problem of developing an efficient heuristic for allocating object fragments in a distributed object oriented database. To accomplish this we created a genetic algorithm which produced more favorable results as compared to the graphical algorithm of Barker and Bhar [2].

Our results show near optimal allocation for those configurations in which the optimal could be computed, improvement over the graphical algorithm and has a linear running time.

*Keywords: distributed object oriented database, object fragments, genetic algorithm, optimal allocation*

## 1 Introduction.

Advances in hardware have encouraged more data intensive applications. These applications include Multimedia, CAD/CAM/CIM and complex financial systems. These applications have processing requirements that are often beyond the capabilities of typical Relational Database Management Systems (RDBMS). RDBMS were developed primarily for managerial and administrative functions and as a result these systems have very little expressive power. The data stored in the majority of these new applications cannot be mapped directly onto the tabular form of a RDBMS. For these new applications, it is important that the database system be able to define its own type of data and operations. These shortcomings of RDBMS have provided an impetus for the development of object-oriented database systems (OODBS). OODBS are based on the object-oriented paradigm

and their main function is to add persistence to objects. One immediate consequence is that OODBS store not only data but also the operations which operate on that data. OODBS therefore support a model describing both the representation and manipulation of data within the database model. Evidence suggests that these types of databases will continue to grow in importance and that they provide superior capabilities for processing complex data.

Distributed databases have been developed to meet the needs of businesses. These enterprises, such as banks and airlines, have thousands of sites located across the world, some sites being geographically far apart. Nevertheless, information services are expected to collect, store, retrieve, process, aggregate and distribute timely information from data generated at these geographically remote and dispersed sites [6]. Gavish and Sheng-Liu [6] suggest that because of this separation of sites, communication delays will be a major problem. Although query optimization can be used to reduce communication costs, it is only optimal with respect to a specified allocation and hence is subject to the efficiency of that allocation. To achieve minimal cost, proper placement of data is required, the object allocation problem (OAP), and this was our major goal in this study.

The remainder of this document is organized as follows. In section 2 we discuss related work on data allocation. We describe the database model we used in section 3 and provide the problem formulation. In this section we describe the terminology used and express the problem goals. Section 4 describes our approach of using genetic algorithms to solve the OAP and we compare our approach with the approaches appearing in the literature as well as to the optimal allocation. Section 5 analyzes the results obtained from our experiments, shows the theoretical soundness of our results and discusses the implications of our results and its position in the wider literature. Section 6 is our conclusion and where we discuss our future work.

## 2 Related Research

### 2.1 File Allocation Problem (FAP)

The problem of allocating files to the nodes of a distributed database system has been an intensively researched problem. An excellent survey appears in Dowdy and Foster [5]. The various works cited differ in solution techniques, data operations considered, evaluation methods, replication factors and network topology. They, however, all consider the problem of how to allocate a set of indivisible files to the nodes of a network to satisfy some cost constraint. The FAP does not however address the correlation of accesses associated with database processing and is of limited interest to our current research.

### 2.2 Database Allocation Problem (DAP)

The majority of the work on database partition allocation has been performed in the context of relational databases. In the distributed database literature the objects to be allocated are usually referred to as partitions. Relational databases are usually partitioned horizontally or vertically. Horizontal partitioning refers to the grouping together of complete tuples (records) while vertical partitioning groups attributes (fields). Partitions are usually designed based on the expected transactions of the system. As far as possible, fragments are designed to match these transactions. An ideal fragment would be when a transaction can be satisfied by one partition. Partitions can also be based on other considerations besides performance, such as security. Apers [1] discusses various approaches and proposes a heuristic. However, due to its complexity, the DAP has never been satisfactorily solved.

### 2.3 Object Allocation Problem (OAP)

The problem of distributing object fragments to nodes of a distributed OODB is the object allocation problem (OAP). The OAP problem differs from DAP because of the additional complexity of distributing objects as opposed to partitions [2], encapsulation and inheritance [9], and class composition.

The first reported work on the OAP problem was Barker and Bhar [2]. They provided a graphical optimization algorithm to create a static allocation satisfying the following goals:

1. The sum of the storage requirements of all the fragments assigned to a particular site must not exceed the total available storage at that site.

2. The total number of non-local references is minimized.

The algorithm they presented performed pairwise site optimization by interchanging fragments at the pair of nodes under consideration. This algorithm, which led to good performance for a network of two nodes, unfortunately demonstrated a sharp decrease in performance as the number of nodes increased.

Related work on distributing object classes to fragments was performed by Bellatreche et al. [3]. Their work not only mapped classes to nodes, but also mapped methods to nodes. Starting with an initial allocation of methods and classes, the algorithm fixed the location of the methods and attempted to find the optimal class allocation. Next, the location of the classes was fixed and an attempt was made to find the optimal method allocation. This alternate optimization sequence was repeated until no improved solution could be found. It is not clear from the literature how this approach compares to Barker and Bhar [2].

## 3 Model Description and Problem Formulation

### 3.1 Model Description

We consider a distributed computer system consisting of  $n$  sites connected by links of fixed bandwidth. Each site has a finite storage capacity and sufficient processing power to process a load of that capacity. We assume a Wide Area Network (WAN) where the nodes of the system may be geographically far apart. This assumption suggests that communication delays will be a major concern. In the formulation of the cost model we assume the communication costs are much larger than the local processing costs at each site. We also assume that the setup transmission costs are negligible. As result we do not include these costs in our formulation. The network is assumed to provide static, reliable and fault free service, therefore we can assume fixed costs between each node pair.

The database is partitioned into a set of  $m$  class fragments with each fragment consisting of a group of objects which are in some way related. All members of a fragment belong to the same class. We assume the existence of a global directory so that all data can be accessed with a constant amount of overhead.

The workload  $W = \{q_0, q_1, \dots, q_r\}$  is set of  $r$  queries where each query invokes a series of methods given by a Method Invocation Sequence  $MIS = \{m_j, m_{j+1}, \dots\}$ .

### 3.2 Intercommunication costs

The intercommunication costs arise when a query originates at a node and executes a method based at the same node or another node. This method will access attributes of its own class which can be located anywhere, and may invoke other methods. The number of times a method accesses an attribute or invokes another method are given by the Method Attribute Utilization (MAU) and Method Method Utilization (MMU). The costs due to accessing attributes arise because the attributes have to be shipped to the site where the method is located, resulting in interfragmentation communication costs. During the invocation of a method, parameter information has to be shipped to the method (parset) and information returned to the invoking method or application (result).

### 3.3 Problem Formulation

We can state our allocation problem in the manner of Barker and Bhar [2] as follows:

Given a set of  $n$  network sites  $S = \{s_1, s_2, \dots, s_n\}$  with respective storage capacities  $C = \{c_1, c_2, \dots, c_n\}$ , a set of  $m$  fragments  $F = \{f_1, f_2, \dots, f_m\}$  where each fragment  $f_i$  is of size  $v_i$  and  $m \geq n$ . Also given a fixed workload  $W$ , and defined  $ASA$ ,  $AFA$  and  $FFR$ ,

find an allocation  $A^*$  such that:

1.  $\forall_i \exists j, \sum_{k|f_k \in F_i} v_k \leq c_j$  (there exists a site where fragment  $F_i$  will be stored)
2. If  $f_k \in F_i$  then  $f_k \notin F_j, \forall i, \forall j, i \neq j$ . (disjointed property)
3.  $\forall k, \exists i, f_k \in F_i$  (Completeness property)
4. Number of accesses between nodes is minimal

(Chapter head:)Preliminary Results

## 4 Evolutionary Approach

The evolutionary approach we take is that of genetic algorithms. Genetic algorithm based search methods [7], are inspired by the perceived mechanisms of natural genetics leading to the survival of the fittest individuals. Genetic algorithms operate on the representation of solutions, *chromosomes*. Associated with each representation is a *fitness* value which determines its goodness with respect to the other members of the population. The higher the fitness value, the more likely the individual will survive in the new generation. As in nature, a *selection* mechanism is the driving force behind the survival of the fittest individuals. The *crossover* mechanism is used to simulate

the recombination of genetic material. Crossover typically works by exchanging portions between strings. Another operator, *mutation* randomly changes a part of the string. This has an analogy with nature, and plays the role of regenerating lost genetic material.

### 4.1 Chromosome

We represent the chromosome as a string of fragment location pairs  $(f_i, s)$  representing a complete allocation of object fragments to network sites. The chromosome has a length equivalent to the number of fragments in the object database. For instance the following chromosome:

$$\{(f_0, s_1), (f_1, s_2), (f_2, s_0)\}$$

represents an allocation of fragment 0 to site 1, fragment 1 to site 2 and fragment 2 to site 0.

### 4.2 Fitness function

To compare our results directly with the graphical algorithm of Barker and Bhar, we utilize their cost evaluation function (Algorithm 4.) [2]. The function which computes the cost between all fragment pairs is given by:

$$\sum_i \sum_j f_i, f_j * DIS(f_i, f_j)$$

### 4.3 Selection

The selection operator selects two parents for mating based on their fitness relative to the remainder of the population. The main goal is for the fittest individuals to be selected for mating more often than less fit individuals. The selection operator we use in this research is tournament selection. In tournament selection [8], a number of individuals are chosen randomly from the population and the best individual from this group is selected as a parent.

### 4.4 Crossover

Single and multi-point crossover define cross points as places between loci where an individual can be split. Uniform crossover [10] generalizes this scheme to make every locus a potential crossover point. A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicate which parent will supply the offspring with genetic material.

### 4.5 Mutation

The mutation operator will choose a gene at random and then randomly change the node value of the gene.

This is equivalent to moving a fragment to another node.

## 5 Experimental results

### 5.1 Test data

We generated the following data based on the information provided by Barker and Bhar [2].

### 5.2 Experiments

We initially compared the algorithm based on its "closeness" to the optimal solution for those cases for which the optimal can be computed. Since the number of possible states is of the form number of  $sites^{fragments}$ , we can only compute the optimal for small values of these parameters. These results appear in Table 1.

Using the Initialization algorithm of Barker and Bhar we also computed the cost improvement over the initial algorithm. The initialization algorithm attempts to place fragments at the sites where applications accessing these fragments are being executed. If there is no space at these sites, then the fragments are placed at nearby sites. The cost improvement is the difference between the cost of this initial allocation and the minimal cost obtained using the genetic algorithm. We compare our improvements directly to those recorded in Barker and Bhar [2] and [4]. The results from this direct comparison appears in Table 2.

We performed a comparison of our algorithm's running time with both an optimal algorithm, and with the reported theoretical running time of the algorithm of Barker and Bhar [2]. We first compare the running time with the exhaustive algorithm. We chose a maximum of 5 sites and 10 data fragments because of the computational expense of running the exhaustive algorithm. The results of this experiment is shown in Table 3. To further analyze the running time of the algorithm, we executed the algorithm with a fixed number of fragments (20) and sites varying from 2-20. The running time is pretty close to linear as can be seen visually in Figure 1. We ran a linear regression model obtaining a  $R^2$  value of 0.999.

We also investigated how the running time of the algorithm varied as the number of fragments increased. We fixed the number of sites at 4, and varied the number of fragments from 40 - 100. The results of this experiment are shown in Figure 2. We also fitted this data to a linear regression model and we obtained a  $R^2$  value of 0.995.

# of sites	# of data fragments	Closeness to optimal
4	10	22
5	10	12
6	10	4
7	10	6
8	9	0
9	8	7
10	7	3

Table 1: Average minimal costs

# of sites	# of fragments	GA	BB
2	100	16.92	10.69
3	100	18.10	7.45
4	100	18.01	9.26
6	100	15.81	4.38
8	100	10.18	1.89
10	100	10.78	0.00
15	100	5.55	2.17
20	100	3.55	1.81

Table 2: Percentage cost improvement compared to graphical algorithm

# of sites	# of data fragments	Exhaustive search	Genetic algorithm
2	10	0.29	1.49
3	10	1.26	2.17
4	10	296.70	2.88
5	10	3,411.58	3.60

Table 3: Average running time of genetic algorithm compared to the exhaustive algorithm (sec)

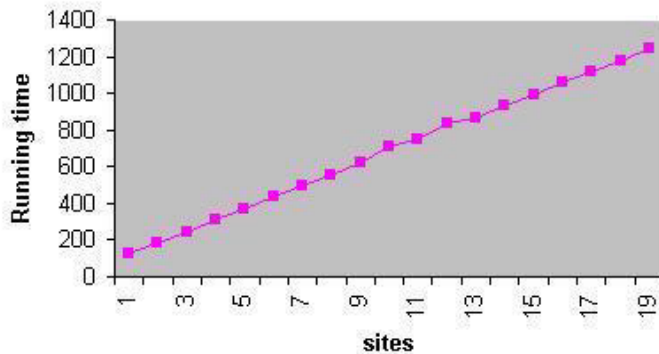


Figure 1:

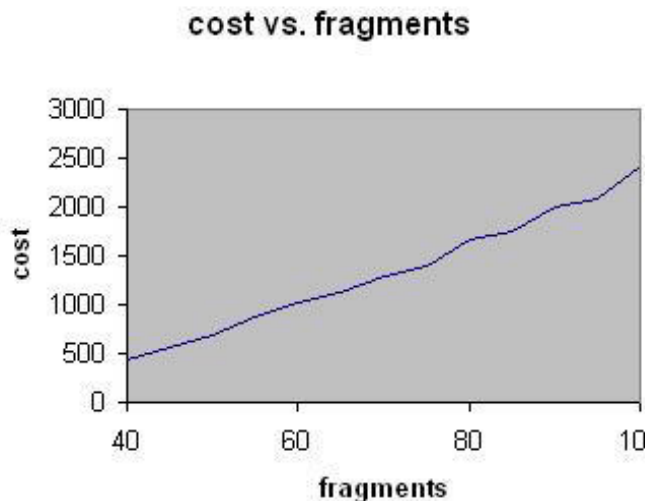


Figure 2:

### 5.3 Analysis of results

In this research, the non-redundant data allocation of object fragments to the sites to a distributed database was addressed. We proposed a solution to this allocation using a genetic algorithm. Initially, the number of fragments and data were chosen so as to allow comparison with the cost of the optimal allocation. Based on the results obtained, the genetic algorithm obtained near optimal results, it was within 22% of the optimal in all cases, and on average it was within 8% of the optimal. These results give some confidence that the algorithm will perform adequately for larger test cases for which obtaining the minimal cost allocation is computationally infeasible. We then made a comparison of the GA against the published results in Barker and Bhar[2] and [4]. Our results showed cost improvements which were from 1.74 % to 11.43 % better than the KL algorithm. The cost improvements however, show a decline for sites in the range 15 - 20. It suggests that the GA might need further tweaking of the parameters.

We also looked at the running time of the algorithms. Table 3 shows the linear increase in the running time of our algorithm as compared to the exhaustive algorithm. The running time of the algorithm exhaustive algorithm is clearly exponential. Since the GA produces results which are close to the optimal, it suggests that it can be used even in those situations where the optimal allocation is feasible. We further investigated the running time of our algorithm to see how the costs vary as the number of sites and number of fragments increase. Figure 1. shows the change in

running time as the number of sites increase. We used a linear regression model to fit the data.. Our impression of linearity was suggested by , and confirmed by a regression  $R^2$  value of 0.99. This running time is much better than the reported theoretical  $O(n^3)$  in number of sites reported in [2] for the running time of the KL algorithm. The running time of the program is also investigated as the number of fragments vary. The results as shown in Figure 2 shows an almost linear increase in cost as the number of fragments increase. We also fitted the data using a linear regression model obtaining a regression  $R^2$  value of 0.995. The theoretical running time of the Barker and Bhar algorithm is  $O(n^3)$  in the number of fragments.

## 6 Conclusion and future work

The allocation of fragments of an object-oriented database to the nodes of a computer network has never been satisfactorily solved. The popularity of the web has fuelled the importance of these databases. It is important that algorithms be in place to perform efficient allocation of fragments to nodes. Attempts in the literature include Barker and Bhar's graphical approach [2], and the approach of Bellatreche and Karlapalem [3]. The graphical algorithm of Barker and Bhar showed some benefits, roughly a 10% performance increase, for a network of two nodes. This improvement is modest at best and deteriorates rapidly as the number of nodes increased Our investigation showed that our algorithm can produce a near optimal allocation and does so efficiently.

There is still a lot of work to be done however. We will attempt to improve the performance of the GA by testing more variations of the parameters. We will also incorporate techniques from more specialized algorithms such as graph partitioning.

Certain assumptions have been made in our model formulation. One is, the existence of sufficient processing power at each node with respect to the data residing there. This may not always be the case and some load balancing maybe required. We will look at the scenario where the processing power at each node is also restricted.

## References

- [1] P.M.G. Apers. Data allocation in distributed database systems. *ACM Trans on Database Systems*, 13(3), September 1988.
- [2] K. Barker and S. Bhar. A graphical approach to allocating class fragments in distributed object-

base systems. *Distributed and Parallel Databases*, 10:207–239, 2001.

- [3] L. Bellatreche and K.Q. Karpalem. Complex methods and class allocation in distributed object-oriented database systems. In *OOIS*, pages 239–256, 1998.
- [4] Subhrajyoti Bhar. *Allocation of class fragments in Distributed Objectbase Systems*. PhD thesis, University of Manitoba, 1996.
- [5] L.W. Dowdy and D.V. Foster. Comparative models of the file allocation problem. *ACM Computing Surveys*, 14(2), June 1982.
- [6] B. Gavish and O. Sheng-Liu. Dynamic file migration in distributed computer systems. *Communications of the ACM*, 13(2), February 1990.
- [7] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.
- [8] D.E. Goldberg and K. Deb. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.
- [9] S. Puro, H. Jain, and Nazareth D. Effective distribution of object-oriented applications. *Communications of the ACM*, 41(8), 1998.
- [10] G. Syswerda. Uniform crossover in genetic algorithms. In *ICGA3*, pages 2–9, 1989.