Improving Data Management in a Distributed Environment

Hassan I. Abdalla
Department of Computer Science
King Saud University, KSA
haabdalla@ksu.edu.sa

Journal of Digital
Information Management

ABSTRACT: *Users of distributed systems often detect performance problems of different types and sources which can sometimes be difficult to trace down to a specific cause due to the complex structure of distributed systems. However, determining the source of the performance problem is highly critical in finding an efficient solution. Managing data fragments on different nodes is one of the major sources of such a problem and is considered as a big challenge facing system developers and designers in a distributed environment.*

This paper contributes in deciding on the most efficient possible allocation alternatives for data partitions in a distributed environment based on the access patterns and the cost data manipulation.

Categories and Subject Descriptors
E 2 [Data Storage Representations]; **H.2.4 [Systems**]; Distributed databases

General Terms: Data Management, Distributed data environment, Data algorithms

Keywords: Distribution, Fragmentation, Query, Distributed Database System

Received: 11 January 2011, Revised 1 March 2011, Accepted 9 March 2011

1. Introduction

Data allocation is a key performance factor for distributed database and data warehouse systems. However, finding optimal solutions for data allocation in a distributed environment is a difficult problem to deal with. This is mainly because many allocation design factors are considered for optimal data distribution. Assuming that the database is properly fragmented, the designer has to decide on the optimal allocation of the fragments to various sites on the network and determine which copy or copies of data to access, where to process and how to route the data.

Typically, users at each site or node have their own set of information requirements. Some of these involve data that is unique to users at a single node. Others require data that is shared among users at multiple nodes. However, to satisfy a user request in a distributed environment, you need to determine where the needed data is located and a strategy that specifies which copy of the data to be accessed and where it will be processed should be identified.

2. Related Work

A distributed database comprises a set of fragments of databases stored at multiple sites that work together and appear as a single database to the user. Each database server in the distributed database is controlled by its local database management system. The objective of data distribution is to meet the information needs of business organizations having different sites with one or more computer systems connected via some communications network.

In a distributed warehouse database system the allocation of data over different sites or nodes of the network is a critical aspect of database design effort. A poor distribution can lead to higher loads and hence higher costs in the nodes or in the communication network, so that the system cannot handle the required set of transactions efficiently.

Fragments allocation problem has been extensively studied in both static and dynamic environments. In a static environment where the access probabilities of nodes to the fragments never change, a static allocation has been proposed prior to the design of a database depending on some static data access patterns. However, in a dynamic environment where these probabilities change over time, the static allocation solution would degrade the database performance. Initial studies on dynamic data allocation give a framework for data redistribution and demonstrate how to perform the redistribution process in a minimum possible time. In [4] a dynamic data allocation algorithm for non-replicated database systems is proposed named optimal algorithm, but no modelling is done to analyze the algorithm. In [14] the threshold algorithm is proposed for dynamic data allocation algorithm which reallocates data with respect to changing data access patterns with special focus on load balancing issues.

Many authors have considered various aspects of the allocation problem, in a variety of contexts. For example, [12] incorporate security considerations into the fragment allocation process [13] consider allocation in the context of multidimensional databases [11] present an algorithm for allocation and replication that adapts to the changing patterns of online requests [2] consider incorporating partitioning into an automatic design framework, and [6] considers incremental allocation and reallocation based on changes in workload.

[15] consider the related problem of distributing the documents of a Web site among the server nodes of a geographically distributed Web server. The problem of replica placement is considered in [7] for networks using a read-one-write-all policy, and in [10] for wide-area systems, while [Buchholz and Buchholz 04] consider it in the context of content delivery networks.

Various approaches have already been adopted to solve the data allocation problem in distributed systems [8], [6], [5]. Some approaches are limited in their theoretical and implementation parts [3], [9]. Other approaches present exponential time of complexity and test their performance on specific types of network connectivity [1].

A major cost in executing queries in a distributed database system is the data transfer cost incurred in transferring fragments accessed by a query from different sites to the site where the query is initiated. The objective of a data allocation algorithm is

to determine the assignment of fragments at different sites so as to minimize the total data transfer cost incurred in executing a set of queries.

In this chapter a new approach for data allocation based on SAGA is going to be presented and analyzed.

3. The Allocation Problem

Fragment allocation is a distribution design technique to improve the system performance by reducing the total query costs. The allocation problem involves finding the optimal distribution of fragments to sites. Optimality can be defined with respect to two measures, cost and performance. In the proposed work it is assumed that the fragments have been determined, and the focus will be on the problem of allocating them in such a way as to minimize the total cost resulting from transmissions generated by user queries.

The objective of the proposed fragment allocation method is to determine which fragments are used by each query being hosted at specific sites such that all queries are satisfied while minimizing the communication cost, processing time, and storage costs, and in the same time not violating storage capacity and processing time constraints.

To describe fragment allocation problem, let's assume that we have a distributed database which is composed of m sites $S = \{S_1, S_2, ..., S_m\}$, where $1 \leq i \leq m$ and each site Si is characterized by memory, CPU and a DBMS and all sites are connected by a communication network and assigned a set of F fragments, where each fragment F_j, is characterized by its size z_j

$$F = \{z_1, z_2, z_3, ..., z_j, ..., z_n\}.$$

Each fragment is requested by at least one of the sites. The site requirements for each fragment are indicated by the query matrix,

$$Q = \begin{vmatrix} q_{1,1} & q_{1,2} & .... & q_{1,n} \\ q_{2,1} & q_{2,2} & .... & q_{2,n} \\ . & . & . & . \\ q_{m,1} & q_{m,2} & .... & q_{m,n} \end{vmatrix}$$

Where $q_{i,j}$ indicates the requirement for fragment j by site i.

A theoretical framework is provided for this problem within the context of relational model that deals with it as a space optimization problem to be solved using SAGA algorithm.

4. Algorithm for Data Allocation

In the proposed SAGA algorithm, GA starts by generating 100 solutions (chromosomes) randomly as the initial population. This population is then used to produce the next generation using the GA operations of selection, crossover and mutation. The newly generated population will contain 100 offspring (new solutions). The process of producing new generations will be repeated 50 times. We will refer to the process of generating 100 initial chromosomes to reproduce 50 generations as Test-1. This operation of Test-1 will be run 100 times to obtain 100 × 50 × 100 chromosomes, accumulating to 500,000 solutions, and each solution has a cost of allocation (fitness value) calculated according to Table 4.2.

SA role in SAGA starts at the parents' selection step of GA. Mainly, SA forces GA to select the parents from a wider space of population by accepting low fitness chromosomes (bad solutions) with the hope to improve solutions in future generations.

We implemented the entire operation of producing 500,000 solutions 4 times, every time using different method:

- First, using GA with random single-point crossover (GA).
- Second, using GA + SA by increasing the temperature from 0 to 100 (SAGA 0-100).
- Third, using GA + SA by decreasing the temperature from 100 to 0 (SAGA 100-0).
- Fourth, using GA + SA by fixing the temperature at 100 (SAGA 100).

Note: The first implementation (GA) is equivalent to SAGA when fixing the temperature at zero. And the fourth implementation (SAGA 100) is equivalent to GA when we select the pair of parents from the entire population (zero fitness).

To obtain accurate and comparable results for the four implementations we used the same input (the initial population) in the four methods. Therefore, we created a repository containing 100 compartments and in each one we randomly generated and saved 100 initial chromosomes. Then we forced each of the four implementations to get its initial population from the generated repository instead of creating it randomly.

For example, in SAGA 0-100 method, to understand the effect of SA on how the GA creates a set of parents, let's assume that we want to produce 25 generations in the whole test and we have 100 chromosomes in the initial population from which GA selected the fittest 25% (25 chromosomes) in the generation number 1. Thus, the remaining 75 chromosomes that represent bad solutions should be included gradually to the set of parents in the subsequent generations controlled by the temperature parameter of SA. The SA will gradually increase the temperature from 0 to 100 at an interval of 4 per generation. At each step of temperature increment the SA forces GA to accept more bad solutions (3 in our example) from the remaining 75%. So, in generation number 2 we will include 3% more chromosomes to the already selected 25% to have a total of 28%. This process will continue until the entire range (100%) of the chromosomes is included by the end of the 25th generation.

Similarly, if we reverse the selection of chromosomes from 100% to 25% then this will represent movement from high temperature to low temperature (100-0). At a high temperature, a large range is sampled, but as the temperature decreases so does the sampling range until the algorithm stops when a freezing condition is met.

5. SAGA Implementation Results

To test the SAGA algorithm we used the same data sample used by [8] as a benchmark and adopted their butterfly network topology (see Fig. 1).

The distributed database used by [8] was containing 15 fragments that need to be allocated over 5 sites, and it was assumed that each site requires specific fragments as presented in Table 1.

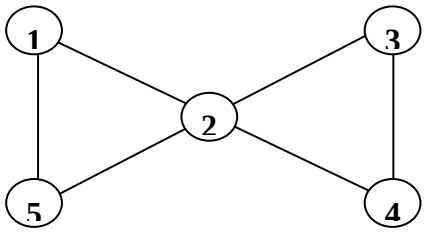

Figure 1. Butterfly Topology

Site	Required Fragments
1	6, 9, 10, 12, 13, 14
2	7, 11
3	3, 4, 5, 6, 10, 12, 13, 14
4	2, 4, 5, 8, 9, 10, 11, 14
5	1, 2, 3, 6, 10, 15

Table 1. Required Fragments

In [8] there was a constraint that each site can host only up to 3 fragments. This constraint has been relaxed in our SAGA approach for data allocation to obtain more feasible solutions. The proposed SAGA allocation model considers only the communication cost and attempts to find an allocation schema that minimizes the total cost function of query processing.

$$\text{i.e. } TC = \sum_{i=1}^{m}\sum_{j=1}^{n} q_{ij} \times t_{ij} \text{ is minimum.}$$

Where $q_{i,j}$ indicates the requirement for fragment j by site i and t_{ij} is the transmission cost for fragment j to site i for the n fragments.

It is also assumed that the cost of data movement from one site to the other is only one unit (see Table 1). According to these assumptions the cost of allocating fragments to sites based on data movements is shown in Table 2.

To see how the communication cost presented in table 2 is calculated let's see for example how the allocation cost for fragment 9 is calculated. If we allocate fragment 9 to site 1, given that fragment 9 was required by sites 1 and 4 (according to table 1). Then the total cost of fragment 9 allocation is composed of two costs:

$$\text{Cost}(9) = \text{Cost}(1, 9) + \text{Cost}(4, 9) = 0 + 2 = 2$$

S/F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	3	3	4	4	3	1	2	2	5	3	2	2	4	1
2	1	2	2	2	2	3	0	1	2	4	1	2	2	3	1
3	2	3	2	1	1	4	1	1	3	5	2	2	2	3	2
4	2	2	3	1	1	5	1	0	2	5	1	3	3	3	2
5	0	2	2	4	4	3	1	2	3	5	3	3	3	5	0

Table 2. Cost of Fragment Allocation to Sites

As fragment 9 was already allocated to site 1, then the cost of requesting fragment 9 from site 1 to site 1 will be zero as it will be local request i.e. Cost(1, 9) = 0. However, the Cost(4, 9) = 2 because the request for fragment 9 from site 4 passes through site 2, i.e. it involves two movements, from site 1 to site 2 and then from site 2 to site 4.

Similarly, if we want to allocate fragment 10 to site 4 then the total cost will be:

$$\text{Cost}(10) = \text{Cost}(1, 10) + \text{Cost}(3, 10) + \text{Cost}(4, 10) + \text{Cost}(5, 10) = 2 + 1 + 0 + 2 = 5$$

Any chromosome (solution) consists of 15 genes (15 fragments) and each gene will take a value between 1 and 5 representing a site number (5 sites). And the total cost for a solution will be calculated according to Table 2. For example, the solution (543435241221135) that cost 23 is calculated in table 3. After executing the aforementioned four implementations we obtained two million solutions (500,000 for each implementation). Fig. 2 illustrates the number of solutions for each cost resulted for the four implementations.

F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	5	4	3	4	3	5	2	4	1	2	2	1	1	3	5
C	0	2	2	1	1	3	0	0	2	4	1	2	2	3	0

Table 3. Calculating the cost for a sample solution

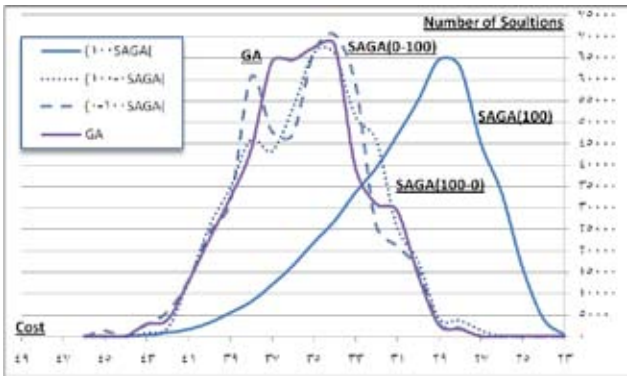

Figure 2. The number of solutions per cost

5.1 Comparing the Results of Different Methods

At the beginning we started with GA and compared its results with that of SAGA 0-100. A little improvement was found when using SAGA 0-100 compared to GA. In the next phase we used SAGA 100-0 which outperformed both GA and SAGA 0-100. However, both of the used SAGA methods didn't significantly enhance GA results and were far below of our expectations. Therefore, we introduced the SAGA 100 method by fixing the temperature at 100 to enforce GA to accept bad solutions with the hope (which come to be true) to improve the obtained solutions as the reproduction process proceeds. If we look at Table 2 and Figure 2, it can be noticed that the number of low cost solutions (like 23, 24 ...etc) are more when using SAGA (both SAGA 100-0 and SAGA 0-100) compared to GA. However, when comparing both SAGA results it is found that SAGA 100-0 is more efficient than SAGA 0-100. This mainly because SAGA 100-0 starts with the entire range of initial solutions and iteratively filters them in the subsequent generations based on the fitness value.

Furtherm ore, SAGA 100 produced much better results (low cost solutions) compared to all other methods, mainly because it used the entire range of initial solutions without filtration.

Additionally, when calculating the average cost for each generation (50 generations), it was found that in SAGA 100 the average cost was noticeably decreasing all through the implementation process. Figure 3 supports the reached conclusions.

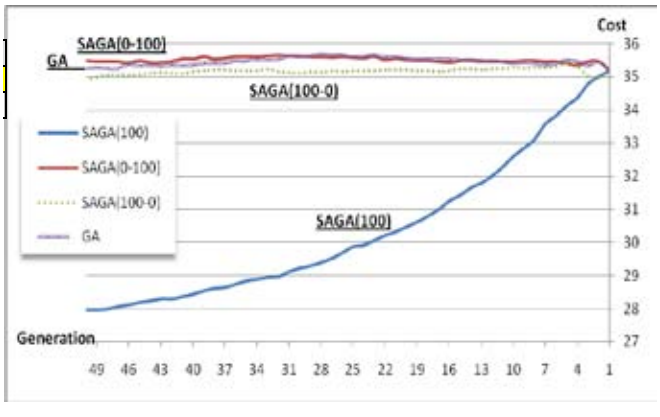

Figure 3. The average cost per generation

5.2 Proposed GA Results VS [8] GA Results

After comparing the different methods of the proposed SAGA algorithm among themselves, I now present a brief comparison of the proposed GA to that proposed by [8] with regard to the best result obtained for the allocation cost.

Both GA algorithms were tested using generational population model where GA saves offspring in a temporary location until the end of a generation where offspring replace the entire current population. [8] used different combinations of crossover operators, however, this study is only interested in the result obtained using single point crossover as it is the crossover method being used in the proposed GA implementation.

The proposed GA implementations started with the same benchmark of [8] considering a problem with 5 sites and 15 fragments, where the fragment size is 1. A fragment matrix was generated in both GA implementations to represent the requirement of each fragment by the different sites (see Table 1).

	Corcoran (GA)	**The proposed (GA)**
Model	**Single Crossover** (Simple)	**Best Result Obtained** (Least cost)
Generational	26	23

Table 4. Our GA Results VS Corcoran's Results

The objective of both GA implementations was to place each fragment to the location that yields the least cost.

Table 4 summarizes the result of the best result obtained by each GA implementation using single point crossover (simple crossover).

6. Conclusion

This paper contributes by allocating data fragments to their optimal location, in a distributed network, based on the access patterns for fragments. A SAGA approach for optimal allocation of data fragments in a distributed environment was proposed, where the mechanism for achieving this optimality relied on knowing the cost involved in moving data fragments from one site to the other.

In SAGA approach, different SAGA methods for data allocation were employed. However, the implementation confirmed that SAGA 100 outperformed all other SAGA and GA methods as it helped us to reach low cost solutions much faster.

When comparing the proposed GA results with the results obtained by [8], it was found that the implementation of the proposed GA has produced better results for the allocation cost compared to the one produced by Corcoran's for a single-point crossover.

However, extending our work using simulated annealing with genetic algorithm (SAGA) has certainly improved the quality of solution for the data allocation problem.

7. Acknowledgment

Authors will like to express their deep appreciation to King Saud University for providing the necessary facilities to accomplish this work.

References

[1] Ahmad, I., Karlapalem, K. Kwok, Y., So, S (2002). Evolutionary Algorithms for Allocating Data in Distributed Database Systems, International Journal of Distributed and Parallel Databases, 11. 5-32, The Netherlands.

[2] Agrawal, S., Narasayya, V., Yang, B (2004). Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design, *In*: Proc. 2004 ACM SIGMOD Int'l Conf. Management of Data, 04.

[3] Apers, P. Data allocation in distributed database systems, *ACM Transactions on Database Systems*, 88.

[4] Brunstroml, A., Leutenegger, S., Simhal, R.(1995). Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database with changing Workloads, *ACM Transactions on Database Systems*, 95.

[5] Buchholz, S., Buchholz, T (2004). Replica Placement in Adaptive Content Distribution Networks," Proc. ACM Symp. Applied Computing, p. 1705-1710.

[6] Chin, A. (2002). Incremental Data Allocation and Reallocation in Distributed Database Systems, Data Warehousing and Web Eng., S. Becker, ed., chapter 7, IRM Press, p. 137-160.

[7] Cook, S., Pachl, J., Pressman, I (2002). The Optimal Location of Replicas in a Network Using a READ-ONE-WRITE-ALL Policy, *Distributed Computing.*.

[8] Corcoran, A., Hale, J (1994). L. C., A Genetic Algorithm for Fragment Allocation in DDSs, ACM.

[9] Huang, Y., Chen, J (2001). Fragment Allocation in Distributed Database Design, *Journal of Information Science and Engineering* 17, 491-506.

[10] Karlsson, M., Karamanolis, C (2004). Choosing Replica Placement Heuristics for Wide-Area Systems," Proc. Int'l Conference Distributed Computing Systems.

[11] Lin, W., Veeravalli, B (2003). An Adaptive Object Allocation and Replication Algorithm in Distributed Databases, *In*: Proc. 23rd Int'l Conf. Distributed Computing Systems Workshops.

[12] Ma, H. Schewe, K.-D. Schewe and M. Kirchberg, (2006). A heuristic approach to vertical fragmentation incorporating query information, *In*: 7th International Baltic Conference on Databases and Information Systems, 3-6 July.

[13] Sto¨hr, T., Ma¨rtens H., Rahm, R (2000). Multi-Dimensional Database Allocation for Parallel Data Warehouses," Proc. 26th Int'l Conf. Very Large Data Bases, p. 273-284.

[14] Ulus, T., Uysal, M (2003). Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems, *Pakistan Journal of Information and Technology* 2 (3) 231-239.

[15] Zhuo, L., Wang, C., Lau, F (2003). Document Replication and Distribution in Extensible Geographically Distributed Web Server, *J. Parallel and Distributed Computing*, 63 (10) 927-944.