International Journal of Database Theory and Application
Vol.10, No.8 (2017), pp.43-56
http://dx.doi.org/10.14257/ijdta.2017.10.8.05

Fragment Allocation and Replication in Distributed Databases

Ali Amiri

Department of MSIS
Spears School of Business
Oklahoma State University
Stillwater, OK, 74078, USA
amiri@okstate.edu

Abstract

We study the problem of designing a distributed database system. We develop optimization models for the problem that deals simultaneously with two major design issues, namely which fragments to replicate, and where to store those fragments and replicas. Given the difficulty of the problem, we propose a solution algorithm based on a new formulation of the problem in which every server is allocated a fragment combination from a set of combinations generated by a randomized greedy heuristic. The results of a computational study show that the algorithm outperforms a standard branch & bound technique for large instances of the problem.

Keywords*: Distributed Database; Fragment Allocation, Fragment Replication, Optimization*

1. Introduction

A distributed database is a single logical database that is spread physically across computers in multiple locations that are connected by a data communications network. The adoption of distributed databases is nurtured by various business conditions such as global nature of business operations and transactions, distribution and autonomy of business units, and telecommunications costs and reliability [7]. A distributed database offers several potential benefits over a centralized database such as improved availability, reliability and performance and lower costs. For those benefits to realize, a distributed database should be properly designed with respects to three important issues: how to partition the database into fragments, which fragments to replicate, and where to store those fragments and replicas [1,11].

As stated in [12], logical database design deals with determining the contents of a database independently of the physical implementation considerations. The guide of the design process is typically the statement of user view requirements/needs in the form of a set of user views [12]. Each view defines the data contents and transactions which are required by a specific job/function that a user view (or group of user views) performs. Each view requires the identification of the database fragments that are needed to process retrieval and update transactions of the corresponding user view. The task of fragmenting the database effectively is a difficult one in itself. However, a variety of approaches exist for fragmenting databases [2,4], and this paper assumes that the fragments have already been identified based on the definition of the user views.

The response to a query in a distributed database environment may require assembling data from several different sites (although with location transparency, the user is unaware of this need). The volume of data transmitted over the communication network depends heavily on the type of the query and the locations of the fragments and their replicas involved in the query. A retrieval query uses only one copy of each fragment in the query; whereas an update query uses every copy of each fragment in the query to maintain

ISSN: 2005-4270 IJDTA

up-to-date data. Besides the way a query is formulated by a user view, query optimization by the database management system affects the query execution efficiency, especially when the query requires fragments located in different sites. In this paper, we design the distributed database in such a way that all the fragments required by a user view are located in the same site. This approach simplifies query optimization and improves response time to users, especially when processing retrieval queries. As this approach allows replication of fragments, an update query still requires updating every copy of the fragments in the queries. While the nonredundant fragment allocation scheme (*i.e.*, storing one copy of each fragment in the database) makes the problem less difficult to solve, it is a very restrictive perspective. Under that scheme, the strategy is first to find a nonredundant solution by ignoring replication and second to apply a heuristic to that solution to decide how to replicate fragments [6,8]. The quality of the final solution obtained using this non-integrated strategy is inferior to the solution of the integrated approach (*i.e.*, which considers fragment replication and allocation simultaneously) that we adopt in our paper [8].

Many studies have considered various aspects of the distributed database design problem in a variety of contexts. A comprehensive review of these studies can be found in [9]. Here, we consider studies that are most related to the topic of our paper. Menon [7] presented new integer programming formulations for the nonredundant version of the fragment allocation problem (where exactly one copy of each fragment exits across all sites). These formulations are solved using the commercial integer programming solver CPLEX [5]. Menon [7] reported computational test results which show that his formulations are more effective than prior formulations using up to 200 fragments and 10 servers.

Hababeh *et al.*, [3] developed a method for grouping the distributed sites into clusters and customizing the database fragments allocation to the clusters and their sites. The method proceeds in three steps: (i) it groups the computing sites into disjoint clusters, (ii) it allocates the fragments to the clusters, and (iii) it allocates the fragments within each cluster to its sites independently of the other clusters and sites. Hababeh *et al.*, [3] reported results of computational tests using a database system with 6 sites and 8 fragments.

Sen *et al.*, [10] studied a general version of the problem where (i) the files/segments have to be clustered and the clusters need to be assigned to a given number of servers whose locations are to be chosen from among a pre-determined set of potential locations, and (ii) query traffic between the users and the servers are routed over a fully connected backbone network. This general problem is solved indirectly by solving first the data partitioning problem and then the segment allocation problem.

In this paper, we present an optimization model for the distributed database design problem (*DDDP*) that deals simultaneously with two major design issues, namely which fragments to replicate, and where to store those fragments and replicas. Again, we assume that the database fragments have already been determined based on the definition of the user views. We develop a mathematical programming model of the *DDDP* which allows a user view to be assigned to a server only if that server is allocated all the fragments required by the user view. In particular, the model considers the objective of minimizing total system cost which is composed of three components: (i) the communication and processing costs of retrieval transactions of the user views, (ii) the communication and processing costs of update transactions of the user views, and (iii) the cost of storing and maintaining the fragments and their replicas on the servers. The decisions to make are (i) how to assign user views to the servers to process their transactions, and (ii) how to replicate the fragments and allocate them to the servers. A solution to the problem should satisfy the following requirements: (i) each user view is assigned to one server to process its transactions, (ii) a user view can be assigned to a server only if all the fragments

needed by the user view transactions are stored on that server, and (iii) the processing capacities of the servers are not exceeded.

Given the difficulty of the distributed database design problem, we propose a solution algorithm based on a new formulation of the problem where every server is allocated a fragment combination from among all combination or from a well generated set of combinations. A combination consists of a group of fragments to be allocated to a server. We run a computational study to study the effectiveness of the solution algorithm. The results show that the proposed algorithm outperforms a standard branch & bound technique based on the first formulation of the problem for large instances of the problem.

The remainder of the paper is organized as follows. Section 2 presents two integer programming formulations of the problem. Two solution procedures are described in Section 3. Computational results are reported in Section 4 while Section 5 concludes.

2. Problem Formulations

Define N the set of user views that need to be assigned to the set M of servers/sites. Define F the set of fragments that need to be allocated to the servers, F_i' the set of fragments needed by user view $i \in N$ in retrieval transactions, and F_i'' the set of fragments needed by user view $i \in N$ in update transactions. Define parameters d_i' to be processing demand of retrieval transactions of user view $i \in N$, d_{ik}'' processing demand of update transactions of fragment $k \in A$ of user view $i \in N$, C_{ij}' communication and processing cost of retrieval transactions of user view $i \in N$ when it is assigned to server $j \in M$, C_{ikj}'' communication and processing cost of update transactions of user view $i \in N$ for fragment $k \in F$ when it stored at server $j \in M$, C_{kj} cost of storing and maintaining fragment $k \in F$ in server $j \in M$, and Q_j processing capacity of server $j \in M$. Define the decision variables as

$$X_{ij} = \begin{cases} 1 & if\ user\ view\ i\ is\ assigned\ to\ server\ j \in M \\ 0 & otherwiese \end{cases}$$

$$Y_{kj} = \begin{cases} 1 & if\ fragment\ k \in F\ is\ stored\ in\ server\ j \in M \\ 0 & otherwiese \end{cases}$$

Given these definitions, the distributed database design problem (*DDDP*) can be formulated as model *DDDP1* below.

DDDP1:

$$min \sum_{i\in N}\sum_{j\in M} C_{ij}' X_{ij} + \sum_{i\in N}\sum_{k\in F_i''}\sum_{j\in M} C_{ikj}'' Y_{kj} + \sum_{j\in M}\sum_{k\in F} C_{kj} Y_{kj} \quad (1)$$

st

$$\sum_{j\in M} X_{ij} = 1 \qquad \forall\, i \in N \quad (2)$$

$$X_{ij} \le Y_{kj} \qquad \forall i \in N, k \in F_i' \cup F_i'', j \in M \quad (3)$$

$$\sum_{i\in N} d_i' X_{ij} + \sum_{i\in N}\sum_{k\in F_i''} d_{ik}'' Y_{kj} \le Q_j \qquad \forall j \in M \quad (4)$$

$$X_{ij}, Y_{kj} \in \{0,1\} \qquad \forall i \in N, k \in F, j \in M \quad (5)$$

The goal is to minimize the total communication and processing cost of user view retrieval and update transactions and cost of setting up and maintaining fragments in the

servers. Constraints (2) ensures that each user view is assigned to one server to handle its retrieval and update transactions. Constraints (3) ensure that if a user view is assigned to a server, then every fragment required by the user view is stored in that server. Constraints (4) limit the total processing demand of the user views assigned to a server to the capacity of the server. The first and second terms in the left hand sides of constraints (4) represent the processing requirements of retrieval and update transactions, respectively. Constraints (5) are restrict the decision variables to be binary.

Since the distributed database design problem (*DDDP1*) is NP-complete, it is extremely difficult to solve it optimally in an acceptable computing time. This difficulty is the direct result of the fact that both the allocation of fragments to the servers and the assignment of user views to the servers have to be handled concurrently. We conducted computational tests that show that the linear programming relaxation of ($DDDP_1$) produces non-integer values of the decision variables and, hence, a standard branch & bound technique using formulation (*DDDP1*) is not likely to generate an optimal solution in reasonable time. We next develop a second formulation of the distributed database design problem that will be used in solving the problem more effectively.

A new formulation for the distributed database design problem can be obtained by viewing it as a covering problem. This formulation stems from the observation that in any feasible solution to the problem, the user views have to be covered by the servers they are assigned to, with a user view being allocated to one server that maintains all the fragments the user view needs in its retrieval and update transactions. As a result, we can solve the problem using all combinations of fragments and assign the "best" combinations to the servers for which the total cost is minimized. Every combination of fragments is a valid configuration that can be allocated to a server.

Using the above insight, the distributed database design problem can now be reformulated in the following way. Define G as the set of all combinations, where a combination is simply a subset of fragments that can be assigned to a server. Define parameter G_i to be the set of combinations which contain all fragments used in processing transactions of user view $i \in N$. Define parameter a_{kg} to be 1 if fragment k is included in combination g. Define a binary variable W_{jg} to be 1 if server j is allocated fragment combination g and 0 if not. Each server needs to be assigned one combination, including the empty combination to leave the possibility that a server may not be used. This can be stated mathematically as

$$\sum_{g \in G} W_{jg} = 1 \qquad \forall\, j \in M$$

Each user view should be assigned to one server whose configuration should include all fragments that the user view needs. Mathematically, this is equivalent to saying

$$X_{ij} \leq \sum_{g \in G_i} W_{jg} \qquad \forall\, i \in N, j \in M$$

The distributed database design problem can now be reformulated as *DDDP2*.

DDDP2:

$$min \sum_{i \in N} \sum_{j \in M} C'_{ij} X_{ij} + \sum_{i \in N} \sum_{j \in M} \sum_{g \in G} \sum_{k \in F''_i : a_{kg} = 1} C''_{ikj} W_{jg} + \sum_{j} \sum_{g} \sum_{k \in F : a_{kj} = 1} C_{kj} W_{jg} \qquad (6)$$

st

$$\sum_{j \in M} X_{ij} = 1 \qquad \forall\, i \in N \qquad (7)$$

$$X_{ij} \leq \sum_{g \in G_i} W_{jg} \qquad \forall\, j \in M, i \in N \qquad (8)$$

$$\sum_{g} W_{jg} = 1 \qquad \forall\, j \in M \qquad (9)$$

$$\sum_{i \in N} d'_i X_{ij} + \sum_{i \in N} \sum_{g \in G} \sum_{k \in F''_i : a_{kg}=1} d''_{ik} W_{jg} \leq Q_j \qquad \forall j \in M \qquad (10)$$

$$X_{ij}, W_{jg} \in \{0,1\} \qquad \forall j \in M, i \in N, g \in G \qquad (11)$$

The goal (6) minimizes the overall cost made of the three components as in (1). Constraints (7) serve the same purpose as constraints (2). Constraints (8) ensure that a user view can be allocated to a server only if the server is assigned a configuration that has all the fragments needed by the user view. Constraints (9) state that each server is allocated a fragment combination. Constraints (10) serve the same purpose as constraints (4). Constraints (11) represent the binary restrictions on the variables.

The following example illustrates the definition of the problem. There are 20 user views, 20 fragments, and 5 servers. The parameters for the retrieval and update transactions are in table 1. All the servers have the same capacity 3663. The storage and maintenance costs for the 20 fragments are {371, 617, 414, 313, 73, 936, 464, 220, 412, 924, 578, 831, 707, 540, 461, 477, 326, 558, 227, 60} for all servers. One possible solution has a total cost of 61706 and consits of assigning user views {6, 7, 9, 10, 12, 20} to server 1, user views {11, 16, 18} to server 2, user views {4, 14, 15, 17} to server 3, user views {2, 3, 5, 19} to server 4, and user views {1, 8, 13} to server 5, and allocationg fragments {1, 2, 3, 5, 6, 7, 8, 9, 11, 13, 14, 15, 18, 19, 20} to server 1, fragments {1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 15, 16, 17, 18, 19, 20} to server 2, fragments {1, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} to server 3, fragments {1, 2, 4, 8, 9, 10, 11, 13, 14, 17, 19, 20} to server 4, and fragments {1, 3, 6, 8, 10, 13, 16, 19} to server 5. The utilizations of the five servers are 96%, 94%, 98%, 55%, and 39%, respectively.

Table 1. Illustrative Example
(a): Retrieval Transaction Parameters

User view	d_i	F_i'	Retrieval cost (C'_{ij})				
			1	2	3	4	5
1	59	8 19	209	156	312	308	151
2	89	2 8 20	411	416	453	256	198
3	86	1 10 13	417	187	416	232	198
4	51	12 17	230	121	129	181	194
5	92	2 10 19	502	530	219	336	291
6	50	5 6	112	164	196	178	286
7	242	3 7 9 15	779	869	972	932	916
8	234	3 6 13 16	1401	1085	1068	914	1001
9	238	1 14 18 20	609	580	1275	1120	734
10	363	6 8 13 14 18 19	1036	2173	1146	1700	997
11	251	1 5 15 20	1306	857	1051	1357	1477
12	313	2 5 6 11 14	802	1356	692	1804	1029
13	229	1 3 6 10	1354	1091	506	831	650
14	534	1 6 7 8 13 16	2391	1593	1903	1216	1347
15	728	11 13 14 15 16 18 19 20	1743	1563	1520	1763	2689
16	750	2 3 6 10 16 17 18 19	3559	3120	3448	4491	4093
17	634	5 7 10 12 14 19 20	3455	2654	1813	3765	2629
18	886	1 4 6 7 8 11 12 17 19 20	4547	3957	5119	4124	4732
19	609	1 4 9 10 11 14 17	2758	3540	3587	2207	1990
20	849	1 2 3 6 11 13 14 18 19 20	1861	4583	4369	3867	1803

(b) Update transaction parameters

User view i	$k(d''_{ik})(C''_{i1k},C''_{i2k},C''_{i3k},C''_{i4k},C''_{i5k})$				
1	19(18)(17,22,24,10,18)				
2	8(16)(17,12,16,11,10)	20(14)(10,10,15,15,11)			
3	10(10)(5,13,7,7,12)	13(14)(15,18,18,11,20)			
4	17(13)(14,17,7,9,9)				
5	2(14)(14,16,20,16,19)	10(15)(11,20,18,8,8)			
6	5(13)(10,18,16,7,9)				
7	3(32)(22,43,28,23,24)	15(35)(30,19,19,24,25)			
8	6(25)(36,35,36,28,34)	16(26)(37,20,16,20,24)			
9	14(28)(16,28,37,25,28)	18(34)(48,26,23,42,49)			
10	8(34)(50,26,40,31,33)	18(29)(23,41,25,30,38)	19(32)(27,24,41,16,34)		
11	1(30)(28,34,32,38,37)	20(34)(47,20,41,39,51)			
12	5(32)(21,27,22,38,31)	14(34)(26,42,26,25,43)			
13	6(28)(37,19,23,23,33)	10(26)(14,14,28,23,24)			
14	7(44)(46,65,65,31,23)	8(41)(46,61,40,50,22)	13(42)(28,50,52,34,25)		
15	13(50)(25,49,29,67,36)	15(49)(26,72,41,34,33)	16(44)(57,35,58,43,52)	19(41)(51,33,40,61,42)	
16	2(50)(36,71,56,45,73)	3(42)(29,53,26,33,39)	6(50)(75,52,56,67,60)	17(46)(34,27,46,41,32)	
17	7(48)(58,35,39,42,26)	10(45)(32,47,38,60,40)	12(42)(39,28,22,32,30)	19(42)(41,30,38,22,32)	
18	4(46)(33,47,46,65,59)	7(42)(58,56,38,62,51)	8(42)(39,42,38,40,51)	11(42)(38,28,45,42,46)	12(45)(28,51,25,44,43)
19	1(41)(57,32,24,34,31)	9(44)(33,50,27,23,54)	10(45)(66,41,30,30,66)	11(49)(50,57,73,34,32)	
20	1(40)(28,28,55,49,45)	13(40)(55,52,25,49,28)	14(43)(37,31,39,57,28)	18(42)(23,55,48,37,61)	20(48)(24,30,49,64,53)

3. CONFIG: A Solution Procedure based on Formulation (*DDDP2*)

While formulation (*DDDP2*) is equivalent to formulation (*DDDP1*), it hides the fact there is an exponential number of configurations, and enumerating all configurations is not a reasonable undertaking. Therefore, we describe a randomized greedy algorithm (*RGA*) to solve the distributed database design problem and, as a by-product, to identify good configurations which can be fed into formulation (*DDDP2*) to solve the problem. A

simple deterministic greedy algorithm involves sequentially selecting a pair of user view and server and allocating the user view to the server. The criterion used in the selection is the net increase in overall cost per unit of demand of the user view. For every user view, we identify the least expensive server to assign the user view to taking into account the fragments that are already allocated to the server. From all pairs of user views and servers, we choose the "best" pair of user view and server that produces the least increase in overall cost per unit of demand of the user view. The heuristic finishes when we assign all user views to the servers. To escape a possible local optimum, we randomize the search by perturbing the costs as follows. If we let Δ_{ij}=Increase in total cost if user view i is allocated to server j, then we use $\tilde{\Delta}_{ij}= (1 + \varepsilon_{ij})\Delta_{ij}$ in selecting the next pair of user view and server, where ε_i is a random number from the uniform distribution U[-0.05, 0.05]. An outline of the algorithm *RGA* is described below.

The following notation is used in describing the algorithm.

C overall cost, initially TC=0
$\bar{N}$ set of un-assigned user views, initially $\bar{N} = N$
$\bar{\bar{N}}$ set of processed user views (*i.e.*, user views assigned to servers), initially $\bar{\bar{N}} = \emptyset$
F_j set of fragments allocated to server $j \in M$, initially $F_j = \emptyset$
$\bar{Q}_j$ remaining capacity of server $j \in M$, initially $\bar{Q}_j = Q_j$

Randomized Greedy Algorithm (*RGA*):

- Let B^* be the best feasible solution found until now.

Repeat 500 times
Step 1: Initialize

- $C = 0,\ \bar{N} = N, \bar{\bar{N}} = \emptyset, F_j = \emptyset,\ \bar{Q}_j = Q_j\ \ \forall j \in M$, B is the solution to construct.

Step 2: User view selection and allocation
While $\bar{N} \neq \emptyset$ do

- For evey pair $(i, j) \in \bar{N} \times M$, compute $\tilde{\Delta}_{ij}= (1 + \varepsilon_{ij})\Delta_{ij}$; the remaining capacity of server j (*i.e.*, R_j) should large enough to handle the additional retrieval and update demand
- Select the pair (i^*, j^*) with the smallest $\tilde{\Delta}_{ij}$. Assign user view i^* to server j^*.
- Update
    - Current solution B.
    - C
    - $\bar{N} = \bar{N} \backslash \{i^*\}$,
    - $\bar{\bar{N}} = \bar{\bar{N}} \cup \{i^*\}$,
    - $F_{j^*} = F_{j^*} \cup F_i' \cup F_i^{"}$
    - $\bar{Q}_{j^*}$

End While

If solution B has a better objective function value than solution B^*, then $B^* = B$.
End Repeat
Retain B^* as the best feasible solution and stop.

During the execution of the algorithm *RGA*, many distinct server configurations are produced; each configuration identifies a set of fragments to store on a server. Such configurations can be fed into formulation (DDDP2) which can be solved with a standard branch & bound algorithm to produce theoretically the best feasible solution to the original problem using those configurations. We refer to this procedure as algorithm *CONFIG*. Given that the subset of configurations employed does not necessarily contain the complete set of best configurations, the optimal solution to the distributed database design problem (*DDDP1*) is not ensured. In order to expedite the branch and bound algorithm, we limit the number of configurations used in formulation (*DDDP2*) to 200 which are part of the "best" solutions generated during the application of *RGA*. The reason

for setting up this limit is that we observed that when we solve model (*DDDP2*) using all the distinct configurations generated by *RGA* the solution quality deteriorates significantly because the solver has to spend a lot of time evaluating all those configurations rather than concentrating on the most promising ones.

We illustrate the performance of the two formulations using the small example described earlier. The solution produced using formulation (*DDDP1*) is optimal with a total cost of 51908 and consits of assigning user views {6, 8, 9, 12, 15, 20} to server 1, user views {7, 11} to server 2, user views {2, 3, 5, 10, 13, 14, 16} to server 3, and user views {1, 4, 17, 18, 19} to server 5, and allocationg fragments {1, 2, 3, 5, 6, 11, 13, 14, 15, 16, 18, 19, 20} to server 1, fragments {1, 3, 5, 7, 9, 15, 20} to server 2, fragments {1, 2, 3, 6, 7, 8, 10, 13, 14, 16, 17, 18, 19, 20} to server 3, and fragments {1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 19, 20} to server 5. The utilizations of the five servers are 99%, 29%, 98%, 0%, and 97%, respectively. The solution produced using algorithm *CONFIG* has a total cost of 52358, corresponding to an optimality gap of only 0.87%, and it consits of assigning user views {6, 8, 9, 12, 15, 20} to server 1, user views {7, 11} to server 2, user views {1, 3, 5, 10, 13, 14, 16} to server 3, and user views {2, 4, 17, 18, 19} to server 5, and allocationg fragments {1, 2, 3, 5, 6, 11, 13, 14, 15, 16, 18, 19, 20} to server 1, fragments {1, 3, 5, 7, 9, 15, 20} to server 2, fragments {1, 2, 3, 6, 7, 8, 10, 13, 14, 16, 17, 18, 19} to server 3, and fragments {1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 17, 19, 20} to server 5. The utilizations of the five servers are 99%, 29%, 95%, 0%, and 99%, respectively.

4. Computational Study and Results

We run a computational study to assess the effectiveness of the proposed algorithm in solving the distributed database design problem (*DDDP*). We coded the algorithm in Visual Studio .Net and run on Intel Core i7 with 3.6 GHz and 8 GB of RAM. We describe here the process of generating the test problems and present and discuss the results.

4.1. Test Problems

We explore the effect of various parameters of the problem, namely number of user views, number of fragments, and number of servers on the effectiveness of the solution procedures using the test problems. We varied the number of user views between 60 and 180, the number of fragments between 10 and 30, and the number of servers between 5 and 25. Ten instances in each category with similar structure were created randomly and solved so as to assess properly the effectiveness of the solution methods on that problem structure. We used in each problem instance three categories of user views of equal sizes: small (*S*), medium (*M*), and large (*L*). Each small user view $i \in N$ requires a number of fragments n_i drawn uniformly from [0.1|*A*|,0.2|*A*|], where |*A*| is the total number of fragments in the database and the retrieval demand d_i' is uniformly distributed between [$20n_i$, $40n_i$]. Each medium user view $i \in N$ requires a number of fragments n_i drawn uniformly from [0.2|*A*|,0.3|*A*|] and the retrieval demand d_i' is uniformly distributed between [$50n_i$, $70n_i$]. Each large user view $i \in N$ requires a number of fragments n_i drawn uniformly from [0.3|*A*|,0.5|*A*|] and the retrieval demand d_i' is uniformly distributed between [$80n_i$, $100n_i$]. For each user view $i \in N$, the number fragments that need to be updated is $0.5n_i$ and the update processing demand d_{ik}'' is drawn uniformly from [10,20] if the user view is small, from [25,35] if the user view is medium, and from [40,50] if the user view is large. The capacity of each server is set to $\lfloor 1.5 * OverallDem/|M| \rfloor$, where *OverallDem* is the total retrieval and update demands of all user views. The communication and processing cost (C_{ij}') of retrieval transactions of user view $i \in N$ when it is assigned to server $j \in M$ is set equal to $d_i' * c_i'$, where c_i' is a uniform random number in [2,6]; the values of C_{ikj}'' are set equal to $d_{ik}'' * c_{ik}''$ where c_{ik}'' is a uniform

number in [1,2]; and the values of C_{kj} were generated from the uniform distribution [50,1000].

4.2. Results and Analysis

The results shown in tables 2-5 are the averages of the ten instances in each category. In Tables 2, 3 and 5, "Iteration Count" represents the number of simplex iterations, "Node Count" represents the number of nodes in the branch & bound tree, and "Gap" is the optimality gap between the best feasible solutions and lower bounds. Table 2 reports the results from solving the distributed database problem with formulation (*DDDP1*) by running CPLEX with the default setting for two hours. As shown in the table, the difficulty of the problem increases in general when its size increases as indicated by the higher optimality gaps. For instance, when the number of servers is 5, the optimal solutions were obtained for all test problems, and when the number of servers is 25, the average gap jumped to 53%. This confirms that it is difficult to solve the problem optimally, or even approximately, using the original formulation (*DDDP1*).

Table 2. Performance of Formulation (*DDDP1*)

$\|N\|$	$\|A\|$	$\|M\|$	Iteration Count	Node Count	Gap(%)
60	20	5	422	0	0.00%
60	20	10	1306742	19313	0.00%
60	20	15	2970267	24282	21.06%
60	20	20	2005198	13856	38.15%
60	20	25	1423374	8608	49.92%
90	20	5	660	0	0.00%
90	20	10	800910	9586	0.00%
90	20	15	1695369	12157	21.39%
90	20	20	1084629	6309	38.71%
90	20	25	892667	3632	51.08%
120	20	5	735	0	0.00%
120	20	10	154629	1131	0.00%
120	20	15	1209725	7582	22.00%
120	20	20	876985	3170	40.83%
120	20	25	619549	1512	53.26%
150	20	5	947	0	0.00%
150	20	10	113634	695	0.00%
150	20	15	955973	4317	21.30%
150	20	20	609966	1756	38.82%
150	20	25	1450336	3852	53.71%
180	20	5	1370	5	0.00%
180	20	10	166073	1008	0.00%
180	20	15	711744	2378	20.65%
180	20	20	733787	1883	40.70%
180	20	25	975622	2108	56.48%

Table 3 reports the results of the tests to evaluate the effectiveness of solving the problem using algorithm *CONFIG* which incorporates the set of fragment combinations produced by the heuristic *RGA* into formulation *DDDP2*. Each combination defines a

subset of fragments that can be allocated to a server. The average number of such combinations used in formulation *DDDP2* is 38.1. The distributed database design problem (*DDDP2*) is computationally less difficult than the problem with formulation (*DDDP1*). The average optimality gap for the solutions obtained using *CONFIG* is only 2.83%. However, it is worth noting that the problem with formulation (*DDDP2*) is still NP-complete, and hence its complexity can grow drastically with problem size. Even though a small number of configurations are used in formulation (*DDDP2*), the effectiveness of this formulation is on average better than that of formulation (*DDDP1*). Indeed, the solutions produced using CONFIG are on average 2.20% cheaper than the solutions produced using formulation (*DDDP1*). Algorithm *CONFIG* based on formulation (*DDDP2*) outperforms formulation (*DDDP1*) when the number servers increases. For instance, when there are 5 servers, both *CONFIG* and CPLEX applied to (*DDPP1*) obtained optimal solution; however, when there are 25 servers, the costs of the solutions produced using CONFIG are on average 6.56% lower than those produced using formulation (*DDDP1*).

Table 3. Performance of Algorithm CONFIG

|N|	|A|	|M|	Configuration Count	Iteration Count	Node Count	Gap(%)	Improvement over *DDDP1* *	Improvement over *RGA* **
60	20	5	1.0	236	0	0.00	0.00	0.00
60	20	10	2.8	22466	626	0.00	-0.66	9.59
60	20	15	40.8	844896	14775	0.00	1.17	8.95
60	20	20	61.4	3561801	31706	1.35	5.20	5.10
60	20	25	53.0	3288801	34676	1.98	8.32	1.65
90	20	5	1.0	541	0	0.00	0.00	1.56
90	20	10	1.4	10955	152	0.00	-0.17	11.70
90	20	15	30.8	3881485	99004	4.00	1.29	13.33
90	20	20	76.2	2277463	22454	3.55	3.13	6.90
90	20	25	72.0	1724473	11958	6.64	8.18	0.53
120	20	5	1.0	511	0	0.00	0.00	4.57
120	20	10	1.0	6555	42	0.00	0.00	13.27
120	20	15	19.4	3111558	49740	3.82	1.61	13.85
120	20	20	73.8	1708248	15910	5.23	2.36	8.52
120	20	25	111.0	1097339	3371	8.38	4.95	-1.03
150	20	5	1.0	531	0	0.00	0.00	4.85
150	20	10	1.0	6300	57	0.00	-0.06	15.09
150	20	15	12.6	3087139	42596	2.91	1.21	15.78
150	20	20	80.8	1520573	9555	6.45	3.08	13.49
150	20	25	118.6	861369	1795	9.82	4.76	1.59
180	20	5	1.0	588	0	0.00	0.00	6.42
180	20	10	1.0	6479	50	0.00	0.00	15.24
180	20	15	9.6	2727864	32561	2.57	0.44	16.55
180	20	20	73.6	1454233	8345	4.96	3.52	14.43
180	20	25	106.4	1486278	2533	9.10	6.59	3.38

* Improvement over *DDDP1* = (*DDDP1* solution - *CONFIG* solution)/*DDDP1* solution

** Improvement over *RGA* = (*RGA* solution - *CONFIG* solution)/*RGA* solution

Table 4 sheds some light on the characteristics of solutions obtained using *CONFIG* such as allocation of cost among its three components, server capacity utilization, number

of fragments stored in the servers, and number of user views allocated to the servers. When the number of servers increases, the contributions of the update and fragment components increase at the expense of the retrieval component. For example, for test problem with 180 user views, when the number of servers is 5, the shares of the retrieval, update and fragment components are 79.9%, 12.3%, and 7.8%, respectively; whereas those numbers become 35.9%. 39.3%, and 24.8% when the number of servers is 25. In addition, when the number of servers increases, server capacity utilization increases, and the number of user views per server decreases; but, the number of fragments per server does not change significantly.

Table 4. Performance of Algorithm CONFIG: Solution Characteristics

$\|N\|$	$\|A\|$	$\|M\|$	Cost			Server Utilization		Number of Frag. per Server		Number of Users per Server	
			Retrieval	Update	Fragmentent	Max	Avg	Max	Avg	Max	Avg
60	20	5	82.2	5.7	12.2	100.0	20.0	20.0	4.0	60.0	12.0
60	20	10	55.5	14.1	30.3	98.4	27.7	20.0	5.9	23.0	6.0
60	20	15	46.7	17.4	36.0	99.8	33.0	17.8	5.6	14.2	4.0
60	20	20	39.3	19.8	40.9	99.9	39.2	15.0	5.7	10.2	3.0
60	20	25	33.4	21.7	44.9	99.9	45.0	13.8	5.7	8.0	2.4
90	20	5	74.5	11.5	14.0	64.1	24.0	20.0	8.0	46.6	18.0
90	20	10	63.3	16.6	20.1	98.9	27.9	20.0	6.0	33.0	9.0
90	20	15	48.4	23.7	27.9	99.8	36.8	18.8	6.9	18.8	6.0
90	20	20	42.0	26.4	31.5	99.9	43.9	16.6	7.0	13.0	4.5
90	20	25	38.0	28.3	33.7	99.9	51.1	14.8	7.1	9.8	3.6
120	20	5	76.7	12.0	11.3	64.0	24.0	20.0	8.0	64.2	24.0
120	20	10	66.2	17.5	16.2	97.9	27.9	20.0	6.0	43.6	12.0
120	20	15	49.0	26.6	24.4	99.9	38.4	19.0	7.5	24.2	8.0
120	20	20	41.4	30.6	28.0	99.9	48.8	18.0	8.2	15.8	6.0
120	20	25	37.1	32.7	30.2	99.9	58.2	15.6	8.5	11.8	4.8
150	20	5	79.2	12.1	8.7	62.9	24.0	20.0	8.0	79.2	30.0
150	20	10	68.9	18.1	13.0	99.4	27.9	20.0	6.0	53.6	15.0
150	20	15	50.7	28.8	20.6	100.0	38.9	20.0	7.7	28.8	10.0
150	20	20	41.8	34.0	24.2	99.9	51.8	18.4	9.0	17.8	7.5
150	20	25	36.5	37.1	26.4	100.0	64.3	16.8	9.7	13.2	6.0
180	20	5	79.9	12.3	7.8	63.3	24.0	20.0	8.0	97.0	36.0
180	20	10	70.2	18.2	11.6	98.5	27.9	20.0	6.0	65.8	18.0
180	20	15	52.2	29.3	18.5	100.0	38.9	19.6	7.7	35.6	12.0
180	20	20	41.0	36.1	22.9	100.0	53.9	18.8	9.5	21.8	9.0
180	20	25	35.9	39.3	24.8	99.9	69.0	16.6	10.7	15.0	7.2

Table 5 shows the computational results of the performance of algorithm *CONFIG* as a results of varying the number of fragments in the database. The table also shows that *CONFIG* produced solutions which have 2.71% lower costs than those produced using formulation (*DDDP1*). This outperformance seems to improve when the number of fragments increases. For instances, when there are 10 fragments in the database, solutions produced using *CONFIG* have on average 1.32% lower costs than solutions produced using formulation (*DDDP1*). However, when the number of fragments increases to 30, solutions produced using CONFIG have on average 4.86% lower costs than produced using formulation (*DDDP1*.

Table 5. Performance of Algorithm CONFIG with Varying Number of Fragments

$\|N\|$	$\|A\|$	$\|M\|$	Configuration Count	Iteration Count	Node Count	Gap(%)	Improvement over *DDDP1* *	Improvement over *RGA* **
60	10	20	38.0	2786878	31759	0.33	1.90	15.47
60	15	20	50.2	2535015	22412	2.24	2.31	15.38
60	20	20	61.4	3561801	31706	1.35	5.20	5.10
60	25	20	45.4	2065879	21433	1.84	2.53	12.85
60	30	20	78.2	3031412	30929	4.47	5.36	13.45
90	10	20	50.0	2377843	25822	2.22	0.90	13.65
90	15	20	68.0	2066529	21943	4.96	0.51	14.75
90	20	20	76.2	2277463	22454	3.55	3.13	6.90
90	25	20	66.4	2030911	18974	4.40	3.52	11.26
90	30	20	74.4	1980880	17583	5.81	5.35	13.35
120	10	20	50.6	1777827	15270	5.00	1.51	13.74
120	15	20	63.4	1566096	12114	7.13	0.25	12.34
120	20	20	73.8	1708248	15910	5.23	2.36	10.20
120	25	20	94.8	1505284	11080	8.39	2.36	16.06
120	30	20	71.0	1629538	13852	6.65	4.92	11.52
150	10	20	42.2	1625033	11205	8.54	0.71	12.16
150	15	20	67.0	1379068	7384	9.27	0.77	13.29
150	20	20	80.8	2432180	17248	6.45	3.08	13.49
150	25	20	80.8	1520573	9555	8.57	3.47	10.68
150	30	20	101.2	1542318	9803	7.28	3.93	11.17
180	10	20	43.0	1672965	9437	7.66	1.58	13.45
180	15	20	70.4	1535671	6677	6.60	0.40	11.43
180	20	20	73.6	1454233	8345	4.96	3.52	12.44
180	25	20	85.4	1277101	8045	8.31	3.52	14.43
180	30	20	76.8	1395915	7862	7.80	4.72	12.50

* Improvement over *DDDP1* = (*DDDP1* solution - *CONFIG* solution)/*DDDP1* solution
** Improvement over *RGA* = (*RGA* solution - *CONFIG* solution)/*RGA* solution

5. Conclusion

A distributed database should be properly designed to benefit from the potential advantages it offers. Effective distribution of the database fragments plays a crucial role in the functioning of the database, affecting both cost and performance. We developed optimization models for the problem that deals simultaneously with two major design issues, namely which fragments to replicate, and where to store those fragments and replicas. Since the problem is very complex, we developed a new formulation of the problem where every server is assigned a fragment combination from a subset of combinations properly produced by a randomized greedy algorithm in which the randomization feature allows the exploration of a larger search space. We conducted an elaborate computational study which shows that the reformulation is far more effective than the original formulation especially for large size instances.

References

[1] H. I. Abdalla, "A New Data Re-allocation Model for Distributed Database Systems", International Journal of Database Theory and Application vol. 5, no. 2, **(2012)**, pp. 45-60.
[2] P. R. Bhuyar, A. D. Gawande and A. B. Deshmukh, "Horizontal Fragmentation Technique in Distributed Database", International Journal of Scientific and Research Publications, vol. 2, no. 5, **(2012)**, pp. 1-7.
[3] I. O. Hababeh, M. Ramachandran and N. Bowring, "A High-Performance Computing Method for Data Allocation in Distributed Database Systems", The Journal of Supercomputing, vol. 39, no. 1, **(2007)**, pp. 3-18.
[4] Y. F. Huang and C. J. Lai, "Integrating Frequent Pattern Clustering and Branch-and-Bound Approaches for Data Partitioning", Information Sciences, vol. 328, **(2016)**, 288-301.
[5] IBM ILOG CPLEX Optimization Studio 12.5, IBM **(2012)**.
[6] K. Karlapalem and N. M. Pun, "Query-Driven Data Allocation Algorithms for Distributed Database Systems", International Conference on Database and Expert Systems Application, Springer Berlin Heidelberg, **(1997)**, pp. 347-356.
[7] S. Menon, "Allocating Fragments in Distributed Databases", IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 7, **(2005)**, 577-585.
[8] M. T. Ö zsu and P. Valduriez, "Principles of Distributed Database Systems", Springer Science & Business Media, **(2011)**.
[9] G. Sen, M. Krishnamoorthy, N. Rangaraj and V. Narayanan, "Facility Location Models to Locate Data in Information Networks: A Literature Review", Annals of Operations Research, vol. 246, no. 1, **(2015)**, pp. 1-36.
[10] G. Sen, M. Krishnamoorthy, N. Rangaraj and V. Narayanan, "Exact Approaches for Static Data Segment Allocation Problem in an Information Network", Computers & Operations Research, vol. 62, **(2015)**, pp. 282-295.
[11] S. Song, "Design of Distributed Database Systems: An Iterative Genetic Algorithm", Journal of Intelligent Information Systems, vol. 45, no. 1, **(2015)**, pp. 29-59.
[12] V. C. Storey and R. C. Goldstein, "A methodology for creating user views in database design", ACM Transactions on Database Systems, vol. 13, no. 3, **(1988)**, pp. 305-338.

Author

Ali Amiri received the MBA and Ph.D. degrees in Management Science/information systems from The Ohio State University, Columbus, OH, in 1988 and 1992, respectively. He is a Professor of Management Science and Information Systems at Oklahoma State University. His research interests include data communications, electronic commerce, data mining, and database management. His papers have appeared in a variety of journals including IEEE Transactions on Communications, European Journal of Operational Research, Computers and Operations Research, INFORMS Journal on Computing, Decision Support Systems, ACM Transactions on Internet Technology, Information Sciences, and Naval Research Logistics.

