Distributed Database Design using Evolutionary Algorithms

Umut Tosun

Abstract: **The performance of a distributed database system depends particularly on the site-allocation of the fragments. Queries access different fragments among the sites, and an originating site exists for each query. A data allocation algorithm should distribute the fragments to minimize the transfer and settlement costs of executing the query plans. The primary cost for a data allocation algorithm is the cost of the data transmission across the network. The data allocation problem in a distributed database is NP-complete, and scalable evolutionary algorithms were developed to minimize the execution costs of the query plans. In this paper, quadratic assignment problem heuristics were designed and implemented for the data allocation problem. The proposed algorithms find near-optimal solutions for the data allocation problem. In addition to the fast ant colony, robust tabu search, and genetic algorithm solutions to this problem, we propose a fast and scalable hybrid genetic multi-start tabu search algorithm that outperforms the other well-known heuristics in terms of execution time and solution quality.**

Index Terms: **Ant colony optimization, distributed database design, hybrid algorithms, robust tabu search.**

I. INTRODUCTION

Fragmentation and data allocation [1] are the two most critical problems when designing distributed databases. Before they are assigned to a site, the relations are mostly partitioned either horizontally or vertically. The replication of fragments is another issue to consider during a design. The memory capacity, communication channels, and processing power are some other design parameters. During the most-frequent queries, the data transferred between sites must be minimized, the locality of the related fragments must be maintained, and the data volume kept on the site must be smaller than the memory size.

A data allocation problem (DAP) is an optimization problem with certain constraints [2]. For instance, the disk I/O speed, parallel query execution, network load, and load balancing of the servers are design parameters that need to be handled. A DAP is an NP-complete problem regardless of these parameters. The most significant factor contributing to the response time of a query is the delay of a data transfer among sites. Therefore, most algorithms deal only with the delay of data to achieve acceptable execution times. A quadratic assignment problem (QAP) has some similarities with a DAP, and also keeps track of the resource locality.

Manuscript received on April 13, 2014.

The author is with the Department of Computer Engineering, Baskent University, Baglica Kampusu, Eskisehir Yolu 20. km 06530 Ankara/TÜRKİYE, Tel: +(90)312 2462099, Fax: (+90)312 246 66 60, email: utosun@baskent.edu.tr.

Digital object identifier 10.1109/JCN.2014.000073

The QAP was first presented by Koopmans and Beckman [3]. A set of *n* facilities and *n* locations is maintained, and the distances between locations are defined. A flow is defined for the amount of supplies to be transferred between each pair of facilities. The problem is assigning the facilities to different locations to minimize the flow between each pair multiplied by the distance between their locations. The QAP can express the dependence of the fragments and sites, where the fragments are considered to be facilities and the sites are considered to be locations. The flow between two facilities is the amount of data transferred between two sites, and the distance between two locations is the cost of sending a data item between the two sites.

The remainder of this paper is organized as follows: Section II describes an overview of both a DAP and a QAP. In Section III, the mathematical formulation behind the modeling of a DAP as a QAP is handled. Section IV presents the algorithms used. The experimental environment is then described in Section V. Finally, Section VI provides some concluding remarks regarding this research.

II. RELATED WORKS

A DAP can be solved using a static or dynamic allocation. Static algorithms exploit the defined prerequisites, whereas dynamic algorithms adapt to the modifications [4]. DAPs and QAPs have been widely studied by the database research community. For example, Ceri and Plagatti proposed a greedy algorithm for redundant and non-redundant data [5]. Ahmad and Karlapalem [6] introduced a query-driven strategy. Adl and Rankoohi [7], transformed a QAP formulation into a DAP, the model of which handles the non-redundant allocation of data with certain capacity constraints. Distributed database management system (DDBMS) queries access several tables and fragments over a network. A query is initialized from a site, and the major portion of the plan execution cost is from the retrieval of fragments from different sites. Data allocation algorithms attempt to assign fragments to sites in such a manner that minimizes the total cost of the data transfer while executing user and/or application queries. DAP algorithms aim to find an optimal fragment-assignment solution while also taking into account the replications, update costs, and average query response times.

Different queries may share the same sub-tasks, and the same queries may be issued from different originating sites. A DDBMS design is a problem with a set of multiple objectives including the efficient usage of computer storage and processing resources, and a minimization of the query response times, while taking care not to violate the constraints regarding the site capacity. It is necessary to model the problem in a way that satisfies all of these criteria. Several algorithms for

data allocation and data fragmentation problems in distributed databases have been proposed in the literature. Techniques based on genetic algorithms (GAs) have been used by Frieder and Siegelmann [8]. However, their formulations do not consider the site capacities or replication of fragments and/or tables to improve the query response times. Ahmad [6] proposed the use of a GA, simulated annealing, and mean field annealing solutions, where non-redundant data (i.e., replications) were not considered. Adl [7] proposed an ant colony heuristic, and modeled the DAP as a QAP; however, the update and replication costs are not handled in this work.

III. SOLUTION FORMULATION WITH QUADRATIC ASSIGNMENT OPTIMIZATION

The data allocation cost can be represented as the sum of direct and indirect transaction-fragment dependencies [7]. A transaction *t* and fragment *f* have a direct dependency if the data from the container site of *f* are transmitted for every execution of *t*. There is an indirect dependency if the data need to be transmitted to a site other than from where the transaction originates. The data allocation cost is expressed as the sum of costs *Cst*1 and *Cst*2, as described in (1).

$$Cst(\Phi) = Cst1(\Phi) + Cst2(\Phi). \tag{1}$$

Here, *Cst*1 is the multiplication of two matrices, *STFR* and *UC*, where *STFR* is the site fragment dependency matrix and *UC* is the unit communication matrix. *UC* holds the network communication costs among the sites. Φ is an *m* element vector and Φ_j represents the site where f_j is stored. Partial cost matrix $PCST1_{n\times m}$ is the cost of fragment f_j to be stored in site s_i. The unit partial cost matrix is represented in (2).

$$pcst1_{ij} = \sum_{q=1}^{n} uc_{iq} \times stfr_{qj}. \tag{2}$$

The unit partial cost $pcst1_{ij}$, for each i and j is calculated, and *Cst*1 is expressed through (3).

$$Cst1(\Phi) = \sum_{j=1}^{m} pcst1_{\Phi_j j}. \tag{3}$$

An inter-fragment dependency matrix (*FRDEP*) is the multiplication of the matrices $QFR_{l\times m\times m}$ and $Q_{l\times m\times m}$ The execution frequencies of the transactions are represented by the matrix *QFR* which is multiplied with matrix *Q* to obtain the *FRDEP* matrix of the inter-fragment dependencies. The indirect transaction-fragment dependency is shown through *Q*. The indirect transaction-fragment dependency cost *Cst*2 is a form of *QAP* and is represented in (4).

$$Cst2(\Phi) = \sum_{j_1=1}^{m}\sum_{j_2=1}^{m} frdep_{j_1 j_2} \times uc_{\Phi_j 1 \Phi_j 2}. \tag{4}$$

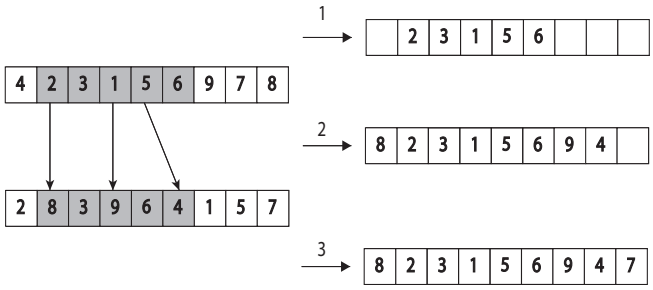

Fig. 1. Stages of PMX crossover.

IV. PROPOSED ALGORITHMS FOR THE DATA ALLOCATION PROBLEM

A. Genetic Algorithm

GAs exploit the selection, crossover, and mutation operations on an initial randomly chosen population. They create new generations, and a fitness function exists to find the best individuals within the population [9]. The termination condition may be defined depending on the total execution time, number of generations produced, or whether no improvements in the average fitness value of the population have been found [10].

A partially mapped crossover (PMX) is used for the GA because it is one of the best performing operators for a QAP solution. The chromosome structure of the solution is shown in Fig. 1. Facilities are placed in an array of locations. The PMX copies a random segment from parent1 to the first child. It looks for elements in that segment of parent2 that have not been copied, starting from the initial crossover point. For each of these elements, e.g., *i*, PMX looks in the offspring to see what element *j* has been copied in its place from parent1, and places *i* into the position occupied by *j* in parent2 because we know that *j* will not be put there. If the position in the offspring occupied by *j* in parent2 has already been filled by *k*, we put *i* in the position occupied by *k* in parent2. The rest of the offsprings can be filled from parent2, and the second child is created in a similar manner [11].

Algorithm 1 Standard ant system

```
Pheromone trail is initialized
while stopping criterion is not met do
  for each ant in the colony do
    Construct a new solution with the current pheromone trail
    Construct an evaluation of the partial solution
  end for
  Update pheromone trail
end while
```

B. Fast Ant System

Ant colony optimization (ACO) was first proposed by Dorigo to solve hard combinatorial problems [12]. It exploits a model based on the real-life cooperation of self-organizing ants. Taillard [13] proposed the fast ant system (FANT) for solving a QAP by incorporating both diversification and intensification. This improves the best solution up to the current execution

Fig. 2. Stages of hybrid genetic multi-start tabu search algorithm.

time of the algorithm in a systematic way by clearing the memory and reducing the weight of the best solution if the process is stagnating. The ant process constructs new solutions by randomly choosing the location of the facilities with a certain probability. The solutions are then improved through a local search and sent to the queen process. While implementing the automatic intensification and diversification, the queen process requires a parameter for managing the traces. The algorithm for a standard ant system is shown in Algorithm 1.

C. Robust Tabu Search

A robust tabu search (RTS) is a well-known optimization algorithm for producing high-quality solutions [14]. It is a variant of a simple TS algorithm. RTS starts with the steepest descent algorithm, and makes up and down movements toward the solution. A tabu list is kept to prevent backward movement for a defined number of moves. RTS tries to evade the local minimum values even when it finds a solution worse than the previous one. It uses adaptive memory, and several of its variants use intensification and diversification to obtain better solutions. RTS has an adaptive tabu list size, which it reduces in order to search near a local minimum, although it can also expand the list size to evade this minimum.

Several aspiration criteria are defined to create exceptions to the restrictions in RTS. RTS has short-term memory and does not maintain the statistics of highly frequent solutions in the way that long-term memory algorithms do. The algorithm generates new permutations by changing the previous allocations of two facilities, which is called a two-way exchange, thereby saving an important amount of execution time. The cost between the old and new permutations is stored in a matrix. Instead of calculating the cost for the whole permutation vector when calculating the cost for a new permutation, these costs are added to the total cost. Although backward movement is forbidden, certain moves are exempt from this rule when they satisfy an aspiration criterion. The tabu list keeps track of the forbidden moves. There is also a parameter called the "total number of failures," which defines the number of unsuccessful iterations for terminating a search for a better solution. The RTS algorithm is similar to Algorithm 2.

D. Hybrid Genetic Multi-Start Tabu Search Algorithm

The hybrid genetic multi-start tabu search algorithm (HG-MTS) is a hybrid of a GA and multi-start RTS. HG-MTS is a two-step algorithm consisting of a seed generation and TS diversification. Fig. 2 shows the stages of this particular algorithm. The TS diversification phase uses the diversification operator of a cooperative parallel tabu search [16]. After choosing a high-quality seed, multi-start TS conducts a stepwise procedure to determine the best diversification toward the solution.

Algorithm 2 Robust tabu search algorithm [14]

```
Authorized: If a move is not tabu, it is authorized.
Aspired: Allow tabu moves if they are decided to be interesting.
Tabu list: A list to forbid reverse move.
Neighbor: Each location in the permutation is considered as neighbor.
RTS (FLOW, DIST, MaxIter, BestPerm,
        MinSize(<n×n/2), MaxSize(<n×n/2),
        Aspiration(>n×n/2)) {
TABU_LIST = {};
CurCost = QAP_Cost(BestPerm);
CurSol = BestPerm;
Delta[i][j] = ComputeDelta();// i = 0..n, j = 0..n
TABU_LIST[i][j] = − (n×i+j); // i = 0...n-1, j = 0...n-1
for (iteration = 1; iteration < MaxIter; iteration++){
      i_retained = infinite;
      MinDelta = infinite;
      Already_Aspired = false;
       for each Neighbor (i, j){
           current1 = TABU_LIST[i][CurSol[j]];
           current2 = TABU_LIST[j][CurSol[i]];
           Authorized = (current1 < iteration) || (current2 < iteration);
           Aspired =
                    (current1 < iteration-Aspiration)||
                    (current2 < iteration-Aspiration)||
                    (CurCost + Delta[i][j] < BestCost);
            if (Aspired && Already_Aspired) ||
                 (Aspired && Delta[i][j] < MinDelta) ||
                 (!Aspired && !Already_Aspired && Delta[i][j] <
                 MinDelta && Authorized){
                 i_retained = i; j_retained = j;
                 MinDelta = Delta[i][j];
                 if (Aspired) Already_Aspired = true;
            }}
      if (i_retained != infinite){
           SWAP(CurSol[i_retained], CurSol[j_retained]);
           CurCost = CurCost + Delta[i_retained][j_retained];
           TABU_LIST[i_retained][CurSol[j_retained]] =
           iteration + getRandom(MinSize, MaxSize);
           TABU_LIST[j_retained][CurSol[i_retained]] =
           iteration + getRandom(MinSize, MaxSize);
           if (CurCost < BestCost)
                BestCost = CurCost;
      }
    UPDATE_MOVE_COSTS(FLOW, DIST, CurSol, Delta,
                                        i, j, i_retained, j_retained);
}}
```

V. EXPERIMENTAL SETUP AND TEST RESULTS

A. Experimental Environment

We tested the proposed algorithms using a number of different experiments. For each test, one of the parameters was varied whereas the others were fixed. The algorithms were tested using the same test data, which were generated based on the rules defined in subsection B. The experiments were performed using a 2.21 GHz AMD Athlon (TM) 64×2 dual processor with 2 GB of RAM and MS Windows 7 (TM) operating system. The implementation language used was C++. The test data were generated according to the experimental environment described by Adl and Rankoohi [7]. The only difference is that we chose a unit cost in range of [0,1]. Our test data generator obtained the number of fragments *m*, number of sites *n*, and other parameters as inputs, and created a random DAP instance.

We chose the fragment size randomly from the range $[c/10, 20 \times c/10]$, where c is a number between 10 and 1000. The random choice of fragments is defined using a constraint because a fragment should be placed at a site with a capacity larger than the fragment size. We chose the site capacities in $[1, 2 \times m/n - 1]$. The sum of the site capacities should be equal to the total fragment size *m*, where *n* is the total number of sites. We assumed that the number of sites *n* is equal to the

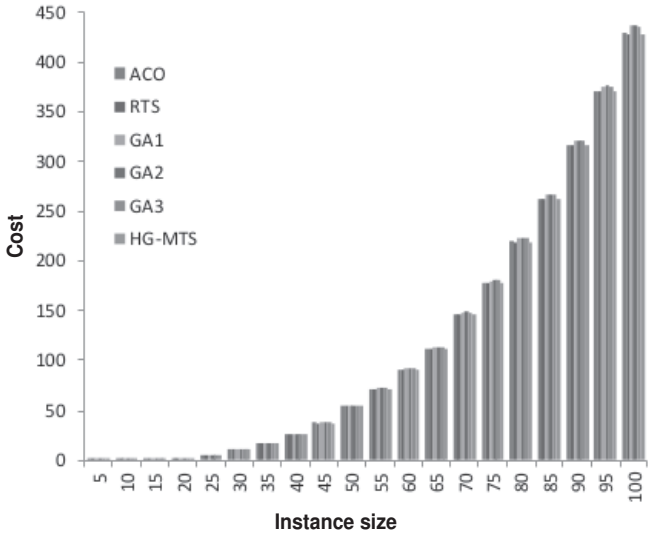

Fig. 3. Cost vs. instance size comparisons of the algorithms.

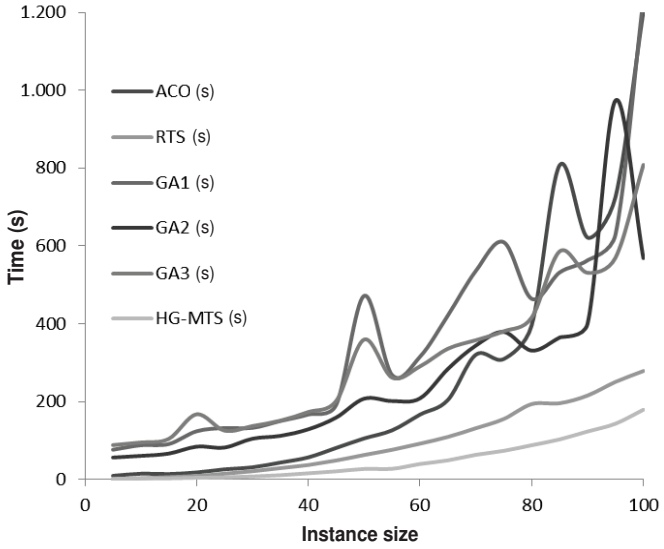

Fig. 4. Time vs. instance size comparisons of the algorithms.

number of fragments *m*. We selected the unit transmission costs as a random number within the range of [0,1]. We generated a random probability request per transaction (*RT*) to allow each transaction to be requested at a site. Transaction fragment dependency is also represented using the probability access per fragment (*AF*). The site fragment frequency matrix, *FREQ*, was determined as the multiplication of probability *RT* and a random frequency of range [1, 1000]. A transaction fragment dependency matrix is generated as a multiplication of *AF* and a uniformly distributed random value in [0, f_j], with f_j being the *j*th fragment.

Finally, the site fragment dependency matrix *STFR* is equal to *FREQ* $\times$ *TRFR*. We define the inter-fragment dependency matrix *FRDEP* as a multiplication of the matrices $QFR_{l\times m\times m}$ and $Q_{l\times m\times m}$, where *QFR* takes into account the execution frequencies of the transactions and *Q* represents the indirect

Table 1. Genetic algorithm performance on DAP-20 instance.

Population	Generation	Cost	Time (s)
250	50	2.655762	4.214
250	100	2.674170	7.777
250	150	2.715429	14.786
250	200	2.673403	15.171
250	250	2.666135	18.124
500	50	2.640902	15.451
500	100	2.660429	19.617
500	150	2.651025	35.112
500	200	2.649761	53.737
500	250	2.649985	76.297
750	50	2.653905	18.796
750	100	2.636911	49.153
750	150	2.646546	63.947
750	200	2.631491	117.159
750	250	2.648080	173.525
1,000	50	2.638806	27.666
1,000	100	2.630268	62.974
1,000	150	2.637986	84.13
1,000	200	2.629396	123.789
1,000	250	2.638621	253.896
1,250	50	2.639087	54.687
1,250	100	2.640549	83.482
1,250	150	2.640518	167.222
1,250	200	2.648000	148.553
1,250	250	2.629725	417.682

Table 2. Genetic algorithm performance on DAP-50 instance.

Population	Generation	Cost	Time (s)
250	50	55.129698	17.581
250	100	55.330358	33.727
250	150	55.235938	55.411
250	200	55.116918	46.914
250	250	55.229595	66.79
500	50	54.946089	28.204
500	100	54.932897	79
500	150	54.833906	86.822
500	200	54.945268	123.973
500	250	54.908336	135.938
750	50	55.121281	41.009
750	100	54.889295	76.333
750	150	54.681993	143.941
750	200	54.650527	183.265
750	250	54.667462	224.648
1,000	50	55.002209	60.278
1,000	100	54.629883	154.437
1,000	150	54.627740	207.555
1,000	200	54.764319	471.978
1,000	250	54.995658	644.301
1,250	50	54.953858	90.265
1,250	100	54.783223	310.599
1,250	150	54.689684	359.574
1,250	200	54.651822	499.853
1,250	250	54.676825	460.665

Table 3. Genetic algorithm performance on DAP-100 instance.

Population	Generation	Cost	Time (s)
250	50	440.608146	48.757
250	100	438.827255	117.298
250	150	439.988411	145.033
250	200	441.213326	181.507
250	250	438.959166	200.907
500	50	438.380273	91.566
500	100	437.361781	174.24
500	150	436.810249	333.716
500	200	438.871504	441.14
500	250	437.437210	493.093
750	50	439.058054	171.855
750	100	436.156120	317.021
750	150	436.641217	618.125
750	200	435.207238	943.319
750	250	435.592737	1,015.88
1,000	50	438.421608	216.33
1,000	100	436.734155	487.567
1,000	150	436.150353	568.726
1,000	200	436.194801	1,236.301
1,000	250	435.735532	1,463.757
1,250	50	438.726232	334.48
1,250	100	436.604972	688.081
1,250	150	434.451806	808.024
1,250	200	435.041686	1,079.677
1,250	250	435.406898	1,455.124

Table 4. Execution time comparison of algorithms for increasing DAP instance sizes.

DAP Size	ACO (s)	RTS (s)	GA1 (s)	GA2 (s)	GA3 (s)	MTS (s)
5	9.26	0.83	76.27	56.11	88.11	1.44
10	14.52	2.73	87.80	60.37	94.91	2.45
15	13.74	5.66	90.76	66.22	104.13	2.65
20	17.91	8.89	123.79	84.13	167.22	4.17
25	25.86	14.52	131.98	81.96	125.30	5.21
30	31.17	20.89	132.46	104.64	137.02	7.38
35	43.31	29.06	150.06	111.87	151.02	10.73
40	56.59	37.05	166.80	128.75	173.21	15.60
45	80.92	48.67	191.93	159.10	202.10	20.80
50	105.33	62.74	471.98	207.56	359.57	26.80
55	126.00	76.07	268.31	201.43	261.71	27.22
60	166.55	91.79	315.31	208.37	290.46	39.56
65	204.35	109.20	421.93	284.08	336.01	48.92
70	320.62	131.54	536.15	344.20	358.03	63.13
75	309.51	155.31	609.77	379.07	380.81	73.41
80	396.18	193.63	464.17	331.17	416.18	87.84
85	807.43	195.80	532.05	364.71	586.21	102.79
90	621.55	215.58	563.15	400.37	531.13	123.19
95	725.93	250.72	629.55	974.24	569.92	143.16
100	1,203.99	278.63	1,236.30	568.73	808.82	179.07

transaction fragment dependency. We used almost the same parameters as Adl and Rankoohi [7] to better understand the performances of these algorithms in the literature.

B. Experimental Results

We performed several tests using a genetic algorithm to set the appropriate parameters. We varied the population size and number of generations to find the optimal running time settings. We performed tests on three *DAP* instances of sizes 20, 50, and 100. In addition, three configuration settings were selected as GA1, GA2, and GA3 after the experiments shown in Tables 1, 2, and 3. GA1 uses a population size of 1,000 and 200 generations. GA2 uses a population size of 1,250 and 200 generations. Finally, GA3 uses a population size of 1,250 and 150 generations. We determined experimentally that these are the best performing parameters. Furthermore, these parameters reflect a performance trade-off among the values because they were chosen in such a way as to minimize the execution time while showing near-optimal solutions.

Table 5. Cost comparison of algorithms for increasing DAP instance sizes (cost value is column$\times 10^6$).

Size	ACO	RTS	GA1	GA2	GA3	HG-MTS
5	0.04	0.04	0.04	0.04	0.04	0.04
10	0.31	0.31	0.32	0.31	0.31	0.31
15	0.98	0.98	0.99	0.98	0.98	0.98
20	2.61	2.61	2.63	2.64	2.64	2.61
25	5.19	5.19	5.25	5.26	5.24	5.19
30	10.27	10.27	10.39	10.42	10.41	10.27
35	16.39	16.39	16.64	16.61	16.66	16.39
40	25.91	25.90	26.28	26.33	26.21	25.92
45	37.28	37.26	37.73	37.80	37.82	37.27
50	53.93	53.89	54.76	54.63	54.69	53.88
55	71.30	71.19	72.72	72.40	72.13	71.21
60	90.35	90.16	91.76	91.49	91.56	90.20
65	112.31	112.13	113.59	113.75	113.84	112.08
70	146.41	146.19	148.48	148.80	148.18	146.15
75	177.90	177.70	180.04	180.75	180.63	177.65
80	219.40	219.26	223.10	222.80	222.96	219.18
85	262.24	261.88	267.04	266.15	266.19	261.99
90	316.11	315.86	320.88	320.93	320.58	315.86
95	370.14	369.92	375.49	375.85	375.29	369.91
100	428.40	428.28	436.19	436.15	434.45	427.98

We used FANT [13] with the parameter $R = 5$ for managing the traces and 20,000 iterations. In addition, we used as the aspiration parameter a maximum of 200,000 failures and $(9 \times n)/10$ and $(11 \times n)/10$ for the lower and upper limits of the tabu list, respectively, where *n* is the instance size. We used a population size of 250, and 50 generations, for the initial phase of HG-MTS. The diversification phase uses 1,000 for the maximum number of failures, and $(n \times n)/10$ and $(11 \times n)/10$ for the lower and upper limits of the tabu list, respectively. These are the optimal parameters reported for both algorithms [13], [14]. After completing the experiments on instances ranging from a size of 5 to a size of 100, it was concluded that HG-MTS outperforms the other algorithms in terms of both time and cost measurements. Only RTS can achieve better results than HG-MTS for a few instances. However, HG-MTS executes more quickly than all of the other methods for all instances as shown in Tables 4, 5, Figs. 3 and 4.

VI. CONCLUSIONS

In this paper, we introduced a new set of quadratic assignment optimization algorithms for designing a distributed database using non-redundant data. We used a well-known genetic algorithm, the fast ant system, and a robust tabu search for the solutions of the data allocation problem. Furthermore, we implemented a more efficient algorithm called HG-MTS by running a modified version of the robust tabu search after operating the genetic algorithm for a number of generations. The main contributions of this work are modeling the problem with using three prevailing algorithms, and the introduction of

the new tabu search based algorithm. In our experiments, the execution times and optimality of the different versions of the quadratic assignment problem algorithms were compared. HG-MTS was shown to outperform the genetic algorithm, fast ant system, and robust tabu search in terms of solution quality and execution times for almost all cases for the data allocation problem. It was observed that the robust tabu search and HG-MTS algorithms outperform the other algorithms particularly when the instance sizes increase. For the smaller instances, it is also obvious that these algorithms obtain the optimal or near-optimal solutions within shorter execution times. Currently, these algorithms consider only one fragment per site. In real life, there is more than one fragment to be considered for a site. Replication is another issue to be dealt with in detail. In the future, we plan to extend the proposed algorithm for the replication and management of multiple fragments for a site. Additionally, the originating sites for the queries and their aspects can be considered to enhance the impact of the proposed algorithms.

REFERENCES

[1] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Springer Publishing Company, 2011.

[2] Z.-J. Lee, S.-F. Su, C.-Y. Lee, and Y.-S. Hung, "A heuristic genetic algorithm for solving resource allocation problems," *Knowledge and Information Systems*, vol. 5, no. 4, pp. 503–511, 2003.

[3] T. C. Koopmans and M. Beckmann, "Assignment problems and the location of economics activities," *Econometrica*, vol. 25, no. 1, pp. 53–76, 1957.

[4] X. Gu, W. Lin, and B. Veeravalli, "Practically realizable efficient data allocation and replication strategies for distributed databases with buffer constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 1001–1013, 2006.

[5] S. Ceri and G. Pelagatti, *Distributed Databases Principles and Systems*, McGraw-Hill, NY: Springer, 1984.

[6] I. Ahmad and K. Karlapalem, "Evolutionary algorithms for allocating data in distributed database systems," *Distributed and Parallel Databases*, vol. 11, no. 1, pp. 5–32, 2002.

[7] R. K. Adl and S. M. T. R. Rankoohi, "A new ant colony optimization based algorithm for data allocation problem in distributed databases," *Knowledge and Information Systems*, vol. 21, no. 3, pp. 349–373, 2009.

[8] O. Frieder, H. T. Siegelmann, "Multiprocessor document allocation: A genetic algorithm approach," *IEEE Trans. Knowl. Data Eng.*, vol. 9, no. 4, pp. 640–642, 1997.

[9] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, MA: Addison-Wesley, 1989.

[10] U. Tosun, T. Dokeroglu, and A. Cosar, "A new robust island parallel genetic algorithm for the quadratic assignment problem," *International J. Production Research*, vol. 51, no. 14., pp. 4117–4133, 2013.

[11] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Springer, 2003.

[12] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating Agents," *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 26, no. 1, pp. 29–41, 1996.

[13] E. D. Taillard, L. M. Gambardella, M. Gendreau, and J. Y. Potvin, "Adaptive memory programming: A unifed iew of meta-heuristics," *European J. Operational Research*, vol. 135, no. 1, pp. 1–16, 2001.

[14] E. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Computing*, vol. 17, no. 4–5, pp. 443-455, 1991.

[15] T. James, C. Rego, and F. Glover, "Multi-start tabu search and diversification strategies for the quadratic assignment problem," *IEEE Trans. Syst., Man, and Cybern., Part A*, vol. 39, no. 3, pp. 579–596, 2009.

[16] T. James, C. Rego, and F. Glover, "A cooperative parallel tabu search algorithm for the QAP," *European J. Operational Research*, vol. 195, no. 3, pp. 810–826, 2009.

Umut Tosun received his B.E. degree from Izmir Institute of Technology, Izmir, Turkey in 2004, his M.E. degree from Bilkent University, Ankara, Turkey in 2007 and his Ph.D. from METU, Ankara, Turkey in 2013, all in Computer Engineering. Between 2004 and 2013, he worked for Bilkent University, ASELSAN Defence, Turkish Telecom, and Siemens Enterprise Communications. He is currently an Assistant Professor in the Department of Computer Engineering at Baskent University. His main interests are parallel and distributed databases, computer networks, and information security.