Heuristic Algorithms for Fragment Allocation in a Distributed Database System

Umut Tosun, Tansel Dokeroglu and Ahmet Cosar

Abstract Communication costs caused by remote access and retrieval of table fragments accessed by queries is the main part execution cost of the distributed database queries. Data Allocation algorithms try to minimize this cost by assigning fragments at or near the sites they may be needed. *Data Allocation Problem* (*DAP*) is known to be NP-Hard and this makes heuristic algorithms desirable for solving this problem. In this study, we design a model based on *Quadratic Assignment Problem* (*QAP*) for the *DAP*. The *QAP* is a well-known problem that has been applied to different problems successfully. We develop a set of heuristic algorithms and compare them with each other through experiments and determine the most efficient one for solving the *DAP* in distributed databases.

Keywords Distributed database design · Fragmentation · Heuristics.

1 Introduction

Although distributed databases (*DDBs*) are attractive for very large datasets, their utilization brings new problems. Designing a *DDB* is one of the most complicated problems in this domain. In addition to the classical centralized database design, fragmentation and data allocation are the new two problems to tackle with [1, 2]. Data allocation problem (*DAP*) is an optimization problem with constraints [3]. Disk drive speed, parallelism of the queries, network traffic, load balancing of servers should be considered during the design. Even without most of these decision parameters

U. Tosun · T. Dokeroglu · A. Cosar (✉)
METU Computer Engineering Department, Ankara, Turkey
e-mail: cosar@ceng.metu.edu.tr

T. Dokeroglu
e-mail: tansel@ceng.metu.edu.tr

U. Tosun
e-mail: tosun@ceng.metu.edu.tr

E. Gelenbe and R. Lent (eds.), *Computer and Information Sciences III*,
DOI: 10.1007/978-1-4471-4594-3_41,

it is clear that *DAP* is an NP-Hard problem. File allocation problem (*FAP*), *DAP*, and *QAP* have some similarities. From the perspective of data transmission, *DAP* is the same with *FAP*. However, the logical and semantic relations of fragments differ from these two problems. *QAP* has more similarities with *DAP* that it also keeps track of the resource locality. The *QAP* is first presented with Koopmans and Beckman [4]. In Sect. 2, we give a brief information about the related works for *FAP*, *DAP*, and QAP. Section 3 explains the design of the solution. Section 4 gives the proposed algorithms and Sect. 5 gives experimental environment and the results respectively. The conclusions are presented in Sect. 6.

2 Related Work

FAP, *DAP*, and *QAP* are extensively studied well-known problems [4, 5]. There are static or dynamic allocation algorithms for the *DAP*. Static algorithms use predefined requirements, whereas dynamic algorithms take modifications into consideration [6]. Ceri and Plagatti presented a greedy algorithm for replicated and non-replicated data allocation design [7]. Bell showed that the *DAP* is NP-Hard [8]. Corcoran and Hale [9] and Frieder and Siegelmann [10] solved the *DAP* with genetic algorithms (*GA*). Ahmad and Karlapalem solved the problem of non-redundant data allocation of fragments in distributed database systems by developing a query driven data allocation approach integrating the query execution strategy with the formulation of the data allocation problem [11]. Adl and Rankoohi addressed prominent issues of non-replicated data allocation in distributed database systems with memory capacity constraint [12]. They took into consideration the query optimization and the integrity enforcement mechanisms in the formulation of the *DAP*. Many other NP-Hard problems are designed by using *QAP* [13–15].

3 Solution Formulation with the *QAP*

The data allocation problem is specified with two kinds of dependencies between transactions and fragments as seen in Fig. 1. The dependency of sites to fragments can be inferred from transaction-fragment and site-transaction dependencies. S represents sites, F represents fragments, T represents transactions in Fig. 1. Similarly, freq is the number of requests for the execution of a transaction at a site, trfr is the direct dependency of a transaction on a fragment, and q is the indirect dependency of a transaction on two fragments.

We formulated the cost function of the data allocation problem as the sum of two costs, direct and indirect transaction-fragment dependencies [12]. The dependency between a transaction t and a fragment f is called direct if there is a data transmission from the site containing f for each execution of t. If there is some data to be transferred from a site different from the originating site of transaction, the dependency is considered as indirect. The total cost of data allocation *Cst* is the sum of two costs *Cst*1 and *Cst*2 (Eq. 1).

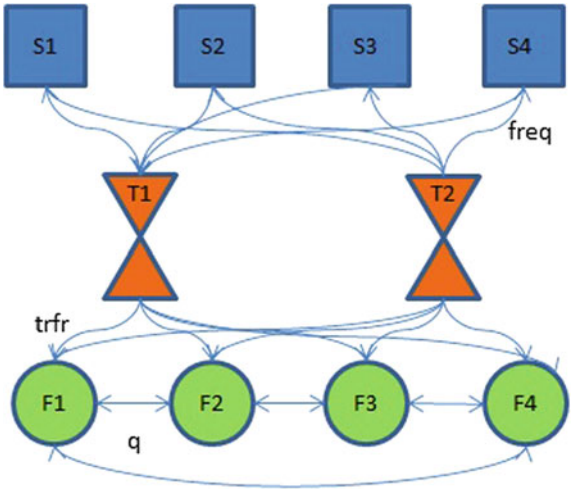

Fig. 1 The dependencies of transactions on fragments and sites on transactions

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

In Eq. 1, Φ represents the m element vector where Φ_j specifies the site to which f_j is allocated. $Cst1$ is represented by the amount of site-fragment dependencies. It is expressed by the multiplication of two matrices *STFR* and *UC*, where *STFR* stores the site fragment dependencies and *UC* stores the unit communication cost among the sites. The cost of storing a fragment f_j in site s_i is represented by partial cost matrix $PCST1_{n\times m}$. The unit partial cost matrix is formulated in Eq. 2.

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

$Cst1$ can be represented as in Eq. 3 after having calculated the unit partial cost $pcst1_{ij}$ for each i and j.

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

The inter-fragment dependency matrix *FRDEP* is defined as the multiplication of the matrices $QFR_{l\times m\times m}$ and $Q_{l\times m\times m}$. The matrix *QFR* denotes the execution frequencies of the transactions. The *FRDEP* matrix representing the inter-fragment dependency is the multiplication of the matrix *QFR* with the matrix *Q*. *Q* represents the indirect transaction fragment dependency. Equation 4 formulates the indirect transaction-fragment dependency $Cst2$. $Cst2$ is a form of the *QAP* as seen in the equation.

$$Cst2(\Phi) = \sum_{j_1=1}^{m} \sum_{j_2=1}^{m} frdep_{j_1 j_2} \times uc_{\Phi_{j1} \Phi_{j2}} \quad (4)$$

4 Proposed Algorithms for the Data Allocation Problem

GAs randomly generate an initial population of solutions, then by applying selection, crossover, and mutation operations repetitively, creates new generations [16]. The individual having the best fitness value is returned as the best solution of the problem [17]. *PMX* crossover is used in the GA. *PMX* copies a random segment from parent1 to the first child. It looks for the elements in that segment of parent2 that have not been copied starting from the initial crossover point. For each of these elements, say i, it looks in the offspring to see what element j has been copied in its place from parent1, *PMX* places i into the position occupied by j in parent2, since we know that we will not be putting j there. If the place occupied by j in parent2 has already been filled in the offspring by k, we put i in the position occupied by k in parent2. The rest of the offspring can be filled from parent2. The second child is created similarly [18].

Dorigo and colleagues proposed *ACO* as a method for solving difficult combinatorial problems [19]. *ACO* is a metaheuristic inspired by the behavior of real ants where individuals cooperate through self-organization. *FANT* is a method to incorporate diversification and intensification strategies [20]. It systematically reinforces the attractiveness of values corresponding to the best solution found so far the search, and on the other hand by clears the memory while giving less weight to the best solution if the process appears to be stagnating.

Metropolis developed a method by simulating the thermodynamic energy level changes in 1953 [21]. With this method, particles exhibit energy levels maximizing the thermodynamic entropy at a given temperature value. The average energy level is proportional to the temperature. This method is called Simulated Annealing (*SA*). Kirkpatrick applied *SA* on computer related problems in 1983 [22]. Many scientists have applied it to different optimization problems since then [23]. If a metal cools down slowly, it turns into a smooth metal because its molecules have entered a crystal structure. This crystal structure shows the minimum energy state, for an optimization problem. If a metal cools down too fast, the metal turns into a rough piece with bumps. These bumps and jagged edges show the local minimums and maximums. In *SA*, each point of the search space represents a state of the physical system, and the function to be minimized is the internal energy of the system in that state. The goal of the algorithm is to bring the system from an initial state to a state with the minimum possible energy.

5 Experimental Setup and Test Results

5.1 Experimental Environment

In each test, one parameter is varied while the others are fixed. The algorithms are tested by the same test data. Data is generated with rules defined in Sect. 5.2. Experiments are performed using a 2.21 GHz AMD Athlon (TM) 64 × 2 dual processor with 2 GB RAM and MS Windows 7 (TM) operating system. Each processing node has 102 buffers, and page size is 10,240 bytes, disk I/O time is 10 ms (per page), available memory is assumed to be sufficient to perform all join operations in main memory and each table is loaded into memory only once. Test data is generated according to the experimental environment of Adl [12]. The only difference is that we choose the unit costs in range [0,1]. Our test data generator gets number of fragments m, number of sites n and other parameters as input and creates a random *DAP* instance. We choose the fragment size randomly from the range $[\frac{c}{10}, 20 \times \frac{c}{10}]$, where c is a number between 10 and 1,000. We choose the site capacities in $[1, 2 \times \frac{m}{n} - 1]$. The sum of the site capacities should be equal to total fragment size m, where n is the total number of sites. We assumed that the number of sites n is equal to number of fragments m. Each fragment size is chosen randomly. We selected the unit transmission costs as a random number in range [0,1]. We generate a random probability of request for each transaction (*RT*) for each transaction to be requested at a site. Transaction fragment dependency is also represented with probability of access for each fragment (*AF*). The site fragment frequency matrix *FREQ* is determined as the multiplication of probability *RT* and a random frequency in range [1, 1,000]. Transaction fragment dependency matrix is generated as the multiplication of *AF* and a uniformly distributed random value in $[0, f_j]$ where f_j is the jth fragment. Finally the site fragment dependency matrix *STFR* is equal to *FREQ* × *TRFR*. We define the inter-fragment dependency matrix *FRDEP* as 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 transaction fragment dependency.

5.2 Experimental Results

We performed several tests over *GA* to set the appropriate parameters. *GA* uses population size 1,000 and number of generations 200. We used the Fast Ant System [20] with parameter $R = 5$ for managing traces and number of iterations as 20,000. *SA* uses 100,000 as number of iterations and 2,750 as number of runs. After completing the experiments on instances ranging from size 5 to 100, it is concluded that *FANT* performs better than *GA* and *SA*. *FANT* executes faster than *GA* and *SA* on all instances as seen in Table 1, Figs. 2 and 3.

Table 1 Comparison of algorithms on *DAP* instances (cost value is column$\times 10^6$) unit

	Cost	Cost	Cost	*DAP* size	Time (s)	Time (s)	Time (s)
DAP size	*ACO*	*GA*	*SA*		*ACO*	*GA*	*SA*
5	0.04	0.04	0.04	5	9.26	76.27	130.29
10	0.31	0.32	0.31	10	14.52	87.80	143.84
15	0.98	0.99	0.98	15	13.74	90.76	214.02
20	2.61	2.63	2.61	20	17.91	123.79	243.30
25	5.19	5.25	5.19	25	25.86	131.98	351.23
30	10.27	10.39	10.27	30	31.17	132.46	461.89
35	16.39	16.64	16.41	35	43.31	150.06	393.73
40	25.91	26.28	26.02	40	56.59	166.80	420.65
45	37.28	37.73	37.40	45	80.92	191.93	437.74
50	53.93	54.76	54.08	50	105.33	471.98	511.40
55	71.30	72.72	71.40	55	126.00	268.31	516.86
60	90.35	91.76	90.50	60	166.55	315.31	828.14
65	112.31	113.59	112.49	65	204.35	421.93	1,090.77
70	146.41	148.48	146.73	70	320.62	536.15	1,303.21
75	177.90	180.04	178.16	75	309.51	609.77	976.97
80	219.40	223.10	219.81	80	396.18	464.17	1,234.48
85	262.24	267.04	262.89	85	807.43	532.05	898.11
90	316.11	320.88	316.81	90	621.55	563.15	1,336.74
95	370.14	375.49	371.14	95	725.93	629.55	1,128.08
100	428.40	436.19	429.10	100	1,203.99	1,236.30	1,389.19

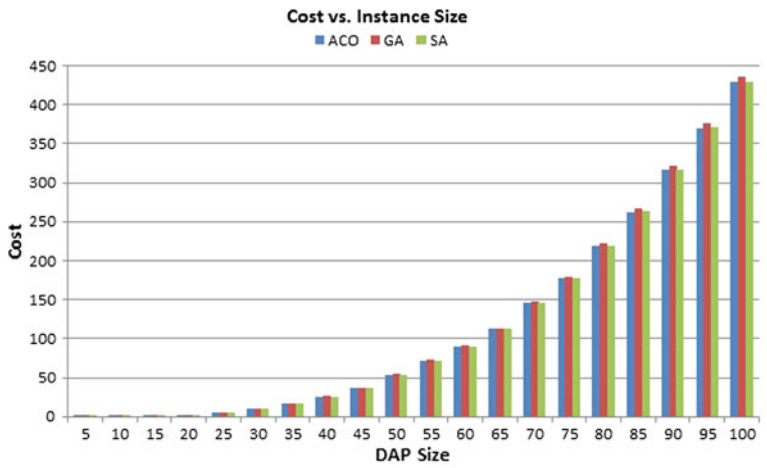

Fig. 2 Cost versus instance size comparisons of the algorithms

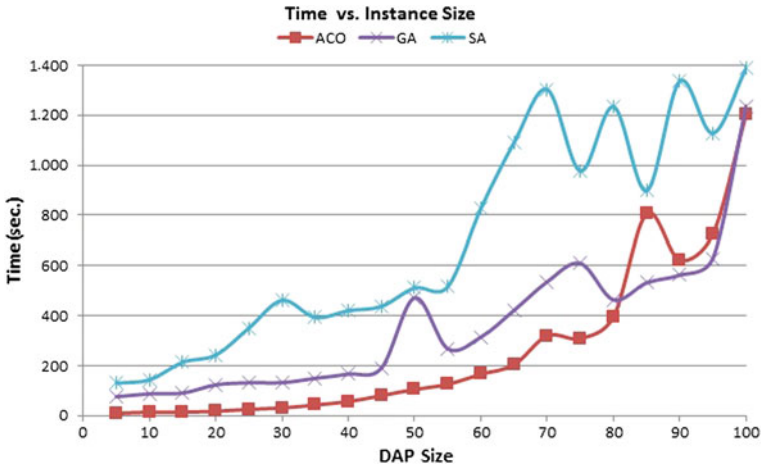

Fig. 3 Time versus instance size comparisons of the algorithms

6 Conclusions and Future Work

We solve the fragment allocation problem in distributed databases by making use of the well known *Quadratic Assignment Problem* solution algorithms. A new set of Genetic, Simulated Annealing, and Fast Ant Colony algorithms are introduced for solving this important problem. In the experiments, the execution times and the quality of the fragment allocation alternatives are investigated. The results are very promising even for very large number of fragments and sites. The model used for deciding the sites where each fragment will be allocated assigns only one fragment to each site. Replication of fragments to multiple sites and assigning multiple fragments to any site have not been considered in this work. As future work, we plan to eliminate these restrictions and develop algorithms that can produce any distributed database schema by allowing replication and horizontal/vertical fragmentation.

References

1. Ozsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 3rd edn, pp. 245–293. Springer (2011)
2. Dokeroglu, T., Cosar, A.: Dynamic programming with ant colony optimization metaheuristic for the optimization of distributed database queries. In: Proceedings of the 26th International Symposium on Computer and Information Sciences (ISCIS), London, Sept 2011
3. Lee, Z., Su, S., Lee, C.: A heuristic genetic algorithm for solving resource allocation problems. Knowl. Inf. Syst. **5**(4), 503–511 (2003)
4. Koopmans, T.C., Beckmann, M.J.: Assignment problems and the location of economics activities. Econometrica **25**, 53–76 (1957)

5. Laning, L.J., Leonard, M.S.: File allocation in a distributed computer communication network. IEEE Trans. Comput. **32**(3), 232–244 (1983)
6. Gu, X., Lin, W.: Practically realizable efficient data allocation and replication strategies for distributed databases with buffer constraints. IEEE Trans. Parallel Distrib. Syst. **17**(9), 1001–1013 (2006)
7. Ceri, S., Pelagatti, G.: Distributed Databases Principles and Systems. McGraw-Hill, New York (1984)
8. Bell, D.A.: Difficult data placement problems. Comput. J. **27**(4), 315–320 (1984)
9. Corcoran, A.L., Hale, J.: A genetic algorithm for fragment allocation in a distributed database system. In: Proceedings of the 1994 ACM Symposium on Applied Computing (SAC 94), pp. 247–250. Phoenix (1994)
10. Frieder, O., Siegelmann, H.T.: Multiprocessor document allocation: a genetic algorithm approach. IEEE Trans. Knowl. Data Eng. **9**(4), 640–642 (1997)
11. Ahmad, I., Karlapalem, K.: Evolutionary algorithms for allocating data in distributed database systems. Distrib. Parallel Databases **11**, 5–32 (2002)
12. Adl, R.K., Rankoohi, S.M.T.R.: A new ant colony optimization based algorithm for data allocation problem in distributed databases. knowl. inf. syst. **25**(1), 349–372 (2009)
13. Dokeroglu, T., Tosun, U., Cosar, A.: Parallel mutation operator for the quadratic assignment problem. In: Proceedings of WIVACE, Italian Workshop on Artificial Life and Evolutionary Computation, Parma, Feb 2012
14. Mamaghani, A.S., Mahi, M., Meybodi, M.R., Moghaddam, M.M.: A novel evolutionary algorithm for solving static data allocation problem in distributed database systems, In: Second International Conference on Network Applications, Protocols and Services, Reviews Booklet, Brussels (2010)
15. Lim, M.H., Yuan, Y., Omatu, S.: Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. Comput. Optim. Appl. **15**(3), 249–268 (2000)
16. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
17. Sevinc, E., Cosar, A.: An evolutionary genetic algorithm for optimization of distributed database queries. Comput. J. **54**(5), 717–725 (2011)
18. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)
19. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. Part B **26**(1), 29 (1996)
20. Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J.Y.: Adaptive memory programming: a unifed view of meta-heuristics. EURO XVI Conference tutorial and research (1998)
21. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087 (1953)
22. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
23. He, X., Gu, Z., Zhu, Y.: Task allocation and optimization of distributed embedded systems with simulated annealing and geometric programming. Comput. J. **53**(7), 1071–1091 (2010)