

- [2] A. Agrawal and R. E. Barlow. A survey of network reliability and domination theory. *Operations Research*, 32(3):478–492, May-June 1984.
- [3] M. O. Ball. Computing network reliability. *Operations Research*, 27(4):823–837, July-August 1979.
- [4] M. O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability*, 35(3):230–239, August 1986.
- [5] P. A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems*, 9(4):596–615, December 1984.
- [6] B. Bhargava and Z. Ruan. Site recovery in replicated distributed database systems. In *6th IEEE Conference on Distributed Computing Systems*, pages 621–627, May 1986.
- [7] P. J. Boland and F. Proschan. The reliability of k out of n systems. *The Annals of Probability*, 11(3):760–764, March 1983.
- [8] P. J. Boland and F. Proschan. Computing the reliability of k out of n systems. In M. S. Abdel-Hameed, editor, *Reliability Theory and Models*, pages 243–255. Academic Press, 1984.
- [9] R. G. Casey. Allocation of copies of a file in an information network. In *AFIPS Conference Proceedings*, pages 617–625, 1972.
- [10] W. W. Chu. Optimal file allocation in a multiple computer system. *IEEE Transactions on Computers*, C-18(10):885–889, Oct 1969.
- [11] K. P. Eswaran. Placement of records in a file and file allocation in a computer network. In *Information Processing (IFIPS)*, 1974.
- [12] B. L. Fox. Discrete optimization via marginal analysis. *Management Science*, 13(3), November 1966.
- [13] D. K. Gifford. Weighted voting for replicated data. In *7th ACM SIGOPS Symposium on Operating Systems Principles*, pages 150–159, December 1979.
- [14] S. Jajodia and D. Mutchler. Dynamic voting. In *ACM SIGMOD Conference*, pages 227–238, 1987.
- [15] S. Jajodia and D. Mutchler. A pessimistic consistency control algorithm for replicated files which achieves high availability. *IEEE Transactions on Software Engineering*, 15(1):39–46, Jan 1989.
- [16] L. J. Laning and M. S. Leonard. File allocation in a distributed computer communication network. *IEEE Transactions on Computers*, C-32(3):232–243, March 1983.
- [17] H. L. Morgan and K. D. Levin. Optimal program and data locations in computer networks. *Communications of ACM*, 20(5):315–322, May 1977.
- [18] L. B. Page and J. E. Perry. A practical implementation of the factoring algorithm for network reliability. *IEEE Transactions on Reliability*, 37(3):259–267, August 1988.
- [19] T. Politof and A. Satyanarayana. Efficient algorithms for reliability analysis of planar networks. *IEEE Transactions on Reliability*, 35(3):252–259, August 1986.
- [20] C. V. Ramamoorthy and B. W. Wah. The isomorphism of simple file allocation. *IEEE Transactions on Computers*, C-32(3):221–231, October 1983.
- [21] L. Resende. Implementation of a factoring algorithm for reliability evaluation of undirected networks. *IEEE Transactions on Reliability*, 37(5):462–467, December 1988.
- [22] R. Tewari. *Robustness in Replicated Databases*. PhD thesis, Rutgers University, 1990.
- [23] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.
- [24] B. W. Wah. File placement on distributed computer systems. *IEEE Computer*, 27(1):23–32, January 1984.

Table II: Accuracy of Solutions Obtained by Proposed Algorithm

	Solution obtained is	
	Optimal	Within 20% of optimal
Prob 1	54	0
Prob 2	NA	NA
Prob 3	54	0
Prob 4	30	24
Prob 5	NA	NA

Note: Entries in the table represent the number of cases of each test problem that fall into the specified category. A total of 54 cases for each test problem were run for each of the availability and the reliability problems. Results for prob3 and prob5 are NA (not available) since the CPU time requirement for exhaustive enumeration of these problem is prohibitive.

Table III: Efficiency of Proposed Algorithm

	Avail. Constraint in Effect (54 cases)	
	Average #of Iterations	Average Efficiency(%)
Prob 1	14	56.25
Prob 2	110	99.99
Prob 3	24	95.31
Prob 4	31	93.95
Prob 5	93	99.93

meration approach. The entries in the table represent the average number of iterations required by the algorithm to solve the stated number of cases and the corresponding average efficiency of the algorithm in solving those cases. The efficiency is calculated as $(1 - (i/2^N)) * 100$, where i is the number of iterations. This formula is derived from the fact that if the the input problem consists of N nodes, then 2^N iterations are required for a complete enumeration.

6 Conclusion

In this paper we have presented an effective integration of the problem of optimally allocating file copies in an DCS while at the same time ensuring mutual consistency of replicated data. This yields a constrained non-linear integer programming problem. An efficient algorithm utilizing the knowledge of the special structure of the problem is proposed for consistency constraints.

The proposed algorithm yielded a solution to all

Table IV: Efficiency of Proposed Algorithm

	Reliab. Constraint in Effect (54 cases)	
	Average #of Iterations	Average Efficiency(%)
Prob 1	12	62.5
Prob 2	132	99.99
Prob 3	32	93.75
Prob 4	33	93.55
Prob 5	103	99.92

our sample problems quite rapidly. The CPU times (on a VAX 8550 computer) for solving problems are less than 1 second for problems with $N \leq 10$, and less than 2 seconds for $N \leq 20$. Thus, our algorithm is well suited for networks with up to twenty nodes even in a real-time mode of operation. Since most network reliability problems can be efficiently solved up-to about fifty nodes [4], we feel that the bottleneck is in the network reliability calculation for larger networks. It is important to note that, fifty nodes is about the size of a large subnet for a public data network if we consider only the gateways. Hence, the proposed algorithm is useful for modeling the subnet of modern networks, where the component computer nodes and communication links are subject to failure, and maintaining mutual consistency of data items is an important consideration.

Acknowledgements

This research was supported by a summer research grant to Raj Tewari from Temple University, and a grant to Nabil Adam from Rutgers, GSM Research Resources Committee and Rutgers University Research Council. We are grateful to the University of California, Berkeley for providing the program *polychain*. The copyright for the program is held by its author, Lucia I. P. Resende. VAX is a trademark of Digital Equipment Corporation.

References

- [1] N.R. Adam and R. Tewari. Regeneration with virtual copies for replicated databases. In *Proceedings of the 11th IEEE International Conference on Distributed Computing Systems*, pages 429–436, May 1991.

- *Otherwise*, the current incumbent solution is the optimal solution. Stop.

Step 3: Perform drop-add phase. The search process always proceeds from the incumbent file allocation.

- The drop part of this phase proceeds as follows: successively dropping a file copy, checking the resulted allocation for feasibility, generating and solving the corresponding LP problem, and replacing the incumbent solution with the new allocation only if the objective function value is lower than that of the incumbent solution.
- The add part of this phase proceeds by successively adding a file copy to the current allocation, checking the resulted allocation for feasibility, generating and solving the corresponding LP problem, and replacing the incumbent solution with the new allocation only if the objective function value is lower than that of the incumbent solution.

If the drop-add phase produces a lower cost allocation, that allocation then replaces the incumbent solution and the next drop-add iteration starts. Otherwise, the incumbent solution is the lowest cost allocation determined by the algorithm and the algorithm stops.

5 Performance Analysis

The proposed algorithm was implemented in FORTRAN and both this program and the *polychain* program ran on a Digital VAX 8550 computer. In order to verify our implementation, we used two test examples [9]: a five node problem, and a nineteen node ARPA problem .

The results of the five experiments conducted using the nineteen node data are summarized in Table I. We observed that, for four out of the five cases, the proposed algorithm results in the same file allocation as the optimal unconstrained file allocation obtained by Casey [9]. Furthermore, the one file allocation that is not found optimally has a cost within one percent of the minimum cost as obtained by Casey. The total costs obtained by our solution procedure and the costs reported by Casey differ by less than one percent.

In order to investigate the performance of the proposed algorithm we experimented with a total of 540 cases of five problems taken from the literature and ten cases of two randomly generated problems. A discussion of the problem set is presented below, followed

Table I: Algorithm Verification Using The Nineteen Node Problem

Update-query Ratio	Casey's Algorithm	Our Algorithm
0.10	2, 10, 14	2, 10, 14
0.20	9, 14	10, 14
0.30	10, 14	10, 14
0.40	10, 12	10, 12
1.00	10	10

by a discussion of the accuracy, efficiency and speed of the algorithm.

5.1 Input problem set

The input problem set consists of seven problems. Five problems, labeled prob1 through prob5, were taken from the network reliability literature. In addition we randomly generated a 50 node problem and a 100 node problem and labeled them as Prob 6 and Prob 7. The link reliabilities for the networks are displayed as the probabilities on the edges. Problem 1, which is made up of 5 nodes and 10 edges, is taken from Casey [9] with node and link reliabilities assumed by us as shown in Figure 2. Problem 2 is taken from Resende [21] and represents the 1974 ARPANET with 21 nodes and 26 edges. Problems 3, 4 and 5 are taken from Page and Perry [18]. For each of these networks, we ran cases varying the update-query ratio from 0.10 to 0.90 in increments of 0.10 and for each update-query ratio we varied both of the required availability and reliability from 0.85 to 0.90. It is worth noting that these ranges are comparable with the ranges that have been used by previous authors (e. g., [9, 24, 16]) in the file allocation literature and in the network reliability literature.

5.2 Accuracy of The Proposed Algorithm

As shown in Table II, our algorithm resulted in the optimal solution for most of the problems considered. In addition, for the rest of the problems, the obtained solutions were within twenty percent of the optimal solution. Results for problems 2 and 5 are not reported due to the very large solution times required for complete enumeration.

5.3 Efficiency of The Proposed Algorithm

Table III and IV demonstrate the efficiency of the proposed algorithm compared to the exhaustive enu-

In the above set of constraints, k varies from 1 to n for equations 4 and 5. The quantity, $A'_i(\mathbf{Y}) = 1 - \prod_{k=1}^n [1 - r_{ik} Y_k u_k]$, and $e'(\mathbf{Y})$ is calculated according to the outline in Section 2.1, by calculating the node reliability and the network reliability, and taking their product to obtain the system reliability.

The formulation in (3) through (10) is a nonlinear integer programming problem. The problem is nonlinear due to constraint (10) which is non-linear in \mathbf{Y} . Constraint (4) arises from the zero-one definition of the control variables, constraint (5) arises from the definition of G'_k , constraint (6) arises from the requirement that the $X_{j,k}$'s being fractions must sum to one for each node j , constraint (7) says that the fraction of file references satisfied by a file copy must be less than or equal to the up-time of the node on which the copy resides, constraint (8) specifies that a node can satisfy a file request only if it has a copy of the file on it (if a node does not have a copy of the file then its $X_{j,k}$ will be zero), constraint (9) is the non-negativity constraint for the $X_{j,k}$ variables. Constraint (10) represents the query availability constraint or the update reliability constraint with e' as the calculated system reliability and e as the required system reliability. Depending on the consistency control algorithm modeled, *either* the availability *or* the reliability constraint will apply, but *not both*.

4 Proposed Algorithm

We present an algorithm to solve the file allocation problem with availability or reliability constraints. This algorithm utilizes the special structure of the problem to solve the resulting linear programming subproblems by inspection.

An examination of the problem formulation reveals that if the Y_j variables are specified, we have a linear programming problem with the $X_{j,k}$ decision variables. Furthermore, this LP has a special structure which can be exploited to obtain efficient solutions, without using a general LP package. The following lemma extends the results presented in Fox [12] for discrete optimization via marginal analysis. Fox considered an LP with only *one* constraint, and proposed an approach where a marginal analysis of the incremental return per additional dollar spent provides an efficient solution. In our problem, we have essentially three constraints for the LP with the $X_{j,k}$ variables. We propose the following lemma which provides an efficient solution procedure for the LP problem generated in the algorithm proposed in the next section:

Lemma 1 Consider the LP problem:

$$\begin{aligned} \text{Min } & c_1 x_1 + c_2 x_2 + \dots + c_n x_n \\ \text{s.t. } & \\ & \sum_{i=1}^n x_i = 1 \\ & x_i \leq b_i \quad \forall i = 1, 2, \dots, n \end{aligned}$$

Further $c_1 < c_2 < \dots < c_n$. A feasible solution to this problem will exist iff $\sum_{i=1}^n (b_i) \geq 1$. If the problem is feasible, an optimal solution can be found by assigning the maximum possible quantities to x_1, x_2, \dots, x_n successively subject to the second constraint, until the x_i sum to 1.

We propose an algorithm which operates by modifying the drop-add heuristic proposed by Wah [20]. Unlike Wah's approach, the approach proposed by Casey could, in the worst case situation, result in exhaustive enumeration. This may take place if the cost function decreases monotonically along all possible paths of the cost graph.

Following is a discussion of the detailed steps.

Step 0: Generate an initial file allocation by assigning a copy of the file to all nodes, i. e. set $Y_j = 1 \forall j$.

Step 1: Check the feasibility of this allocation by calculating the availability or quorum reliability (as applicable).

- If this allocation is feasible generate the corresponding LP subproblem for the $X_{j,k}$'s and solve it using the efficient solution procedure discussed in Lemma 1. The value of the corresponding objective function represents the minimum cost allocation obtained so far. Proceed to Step 2.
- Otherwise the problem has no feasible solution. Stop.

Step 2: Perform an initial drop-phase, where each copy of the file is dropped in turn and the feasibility of the resulted file allocation is checked (as discussed in Step 1); if found feasible the corresponding LP is generated and solved, and value of the objective function is compared to the previous minimum cost, and:

- If the initial drop-phase produces a lower cost allocation, store this as the incumbent minimum cost allocation and proceed to Step 3.

2.2 Node Availability

We now turn our attention to the second category of consistency control algorithms that can handle *site failures only*. ROWAA and Available Copies are examples of such algorithms. According to these algorithms, the first available copy of a file is used to satisfy a user query. Due to the communication costs, the local copy is preferred over a remote copy. The availability of a given file can be defined as the probability that a copy of the file is accessible when requested by users.

We define r_{ik} , which is the probability of successful communication over some network path between nodes i and k . Thus, $r_{ii} = 1$, by definition. It is interesting to notice that r_{ik} and a_{ij} are different due to the replication of file j at nodes other than node i . The parameters r_{ik} are calculated once for every network, for every pair of nodes i and k .

In this study we consider a realistic setting where nodes may fail. We define the following new terms:

$$Y_k = \begin{cases} 1 & \text{if node } k \text{ contains a copy of the file} \\ 0 & \text{otherwise} \end{cases}$$

and

$$u_k = \begin{cases} 1 & \text{if node } k \text{ is operational} \\ 0 & \text{otherwise} \end{cases}$$

Substituting Y_k for x_{ij} the file allocation decision variables can be represented by the vector $\mathbf{Y} = (Y_1, Y_2, \dots, Y_n)$ and we have:

$$A'_i(\mathbf{Y}) = 1 - \prod_{k=1}^n [1 - r_{ik} Y_k u_k] \quad (1)$$

The availability measure is incorporated into the file allocation problem as a set of constraints of the form [16]:

$$\frac{1}{N} \sum_{i \in S} (A'_i(\mathbf{Y})) \geq a \quad (2)$$

where S is the set of all network nodes having demand for the file, N is the number of nodes in the set S and a is the user specified minimum availability requirement for the file.

3 Formulation of The Extended File Allocation Problem

We first define the problem parameters and decision variables:

Input Parameters

σ_k = storage cost of locating a copy of the file at node k (cost/time-period).

λ_j = volume of query traffic originating from node j (megabits/time-period).

ψ_j = volume of update traffic originating from node j (megabits/time-period).

d_{jk} = unit communication cost from node j to node k for a query transaction.

c_{jk} = unit communication cost from node j to node k for an update transaction.

n = number of nodes in the network.

u_k = probability of node k being operational.

r_{jk} = probability of successful communication over some network path between nodes j and k .

a = desired system availability.

e = desired system reliability.

Decision Variables

X_{jk} = fraction of file requests of node j answered by file copy at node k .

$$Y_k = \begin{cases} 1 & \text{if node } k \text{ contains a copy of the file} \\ 0 & \text{otherwise} \end{cases}$$

The objective function can now be written as:
Min

$$C(\mathbf{X}, \mathbf{Y}) = \sum_{j=1}^n \sum_{k=1}^n \lambda_j d_{jk} X_{jk} + \sum_{k=1}^n G'_k Y_k \quad (3)$$

and the complete set of constraints are as follows:

$$Y_k = 0 \text{ or } 1 \text{ (integer)} \quad (4)$$

$$G'_k = \sigma_k + \sum_{j=1}^n \psi_j c_{jk} u_k r_{jk} \quad (5)$$

$$\sum_{k=1}^n X_{jk} = 1, \quad j = 1, 2, \dots, n \quad (6)$$

$$X_{jk} \leq u_k \cdot r_{jk} \quad \forall j, k \quad (7)$$

$$X_{jk} \leq Y_k \quad \forall j, k \quad (8)$$

$$X_{jk} \geq 0 \quad \forall j, k \quad (9)$$

$$\frac{1}{n} \sum_{j=1}^n (A'_i(\mathbf{Y})) \geq a \quad \text{or} \quad e'(\mathbf{Y}) \geq e \quad (10)$$

- The probability that the K or more nodes are able to communicate, i. e. have an operational path between them. This is referred to as the *network reliability*.

In this paper we assume that node failures are independent of end-to-end path failures. Given the current state of wide area computer networks, we believe that this assumption is justified, since such networks typically include alternate paths among nodes.

Following is our proposal for obtaining an approximation of both components of the quorum reliability.

2.1.1 Node Reliability

Node reliability is equivalent to the probability that k or more of the nodes are operational and contain a copy of the file. This problem can be formulated as the problem of determining the reliability of k -out-of- N system.

According to Boland and Proschan [7], if the average (over all nodes) reliability of the system, p , is high then the reliability of the k -out-of- N system is closely approximated by the binomial distribution with parameters p and N . It is further shown in [8] that the approximation is sharp for high node reliabilities ($p > 0.9$) thereby providing a good lower bound.

In [8] it was observed that the calculation of the reliability function $h_k(\mathbf{p})$ becomes easier as the number of distinct component values in \mathbf{p} decreases. Consider for example a system with eight components. If we have $\mathbf{p} = (p_1, p_2, \dots, p_8)$ with each element being $\geq \frac{k-1}{N-1}$, then we can easily determine lower bounds for the reliability calculation.

In the context of our problem, we will be concerned with systems where the property: $p_i \geq \frac{k-1}{N-1}, \forall i$ is satisfied. since voting (or a variant of voting) method is being used. Such methods always look for a majority of the current and operational copies in each partition, hence k is in the range of 1/2 of N . Consequently, $\frac{k-1}{N-1}$ is in the neighborhood of .5 and we would then have $p_i \geq \frac{k-1}{N-1}$.

We use the above result for obtaining the lower bound on the node reliability component of the quorum reliability, and adopt this approximation rather than calculating upper and lower bounds by enumerating all path sets and cut sets, which is computationally infeasible for large problems.

2.1.2 Network Reliability

Surveys of network reliability as well as efficient algorithms to calculate it are provided in [2, 19]. In [19] it

was stated that “the reliability problem for a general network has been shown to be inherently difficult and very likely no efficient algorithm can be constructed for its solution.”

For the purpose of calculating the network reliability, we assume that nodes fail independently of end-to-end paths. Our problem then reduces to the calculation of the k -terminal reliability, given that k is the size of the quorum. We are interested in obtaining a point estimate of the network reliability. A discussion of the computational complexity of this problem can be found in [4], where it is stated that the point estimate reliability analysis problems are all NP-hard for the k -terminal network reliability measure.

Graph reductions have been described in [2, 3], and a factoring algorithm has been proposed to exploit the structure of the network to obtain reductions in the graph, thereby simplifying the reliability calculations.

In [21] a FORTRAN implementation (referred to as *polychain*) of a factoring algorithm for reliability evaluation of undirected networks is presented. This program works by reducing the size of the input graph through reliability-preserving transformations. Three types of reductions are considered: parallel reduction, series reduction, and degree-2 reduction. For a given edge, e , the p_e and $q_e = 1 - p_e$ denote the edge reliability and the edge failure probability respectively. In parallel reduction, for example, two parallel edges that connect the same two nodes, e_a and e_b are replaced by a single node e_c , whose edge reliability = $1 - q_a q_b$. For a given graph, G , the algorithm goes recursively through the following steps:

- Reduce (G).
- Select edge e to pivot. That is, chose an edge whose removal does not disconnect the graph.
- Factor (G_e) and ($G - e$), where (G_e) is the graph G considering that edge e is working and ($G - e$) is the graph G considering that edge e is not working.

The the k -terminal reliability, $R_k(G)$, is computed by repeated applications of the following decomposition,

$$R_k(G) = p_e R_k(G_e) + (1 - p_e) R_k(G - e)$$

We have used this program to compute the network reliability, for a discussion of the implementation and further details on the underlying algorithms see [21].

tion problem (GFAP), as opposed to an unconstrained *simple file allocation problem* (SFAP).

Ramamoorthy and Wah [20] proved that there is an isomorphism between the simple file allocation problem and the single commodity warehouse location problem. Solution techniques proposed for one problem can therefore be used to solve the other problem. Branch and bound, and add-drop heuristics have been proposed for the efficient solution of both problems.

The main contribution of our paper is to incorporate the behavior of consistency control algorithms that ensure mutual consistency of replicated data, in the file allocation problem for distributed computing systems. This is accomplished by adding probabilistic reliability and availability constraints to the original model and modifying the objective function accordingly. We propose an efficient algorithm for the resulting non-linear integer programming model. Our model takes into account failures of computers on which the files reside, as well as failures of communication links of the computer network. In previous work computers (nodes) have been assumed completely reliable, and only communication links were subject to failure.

The rest of the paper is organized as follows. In the next Section, consistency considerations are introduced and modeled as reliability or availability constraints depending on the type of consistency control algorithm being used. Section 3 states the extended file allocation problem including the consistency constraints. Section 4 describes a solution algorithm. Section 5 presents an implementation of our algorithm, followed by a performance analysis of the algorithm in Section 6 in terms of the algorithms accuracy, efficiency, and execution speed.

2 Consistency in Distributed Systems

Distributed computing systems having multiple copies of data items may have strict *consistency* requirements for each data item. That is, all copies of a data object should have *the same value*, ensuring that answers to user queries have a consistent value across the distributed system. To illustrate, let us consider the following examples. In a banking environment, an account balance should have the same amount whether it is queried from New York or San Francisco. In an airline environment, the data object may be a reservation file containing all seat assignments from New York to San Francisco to provide a consistent answer to user queries issued from any airline reservation center. In a military environment, the data object may be a radar tracking file used to home

in on a target based on multiple sensors, where it is absolutely essential to have a consistent picture of the target.

Consistency control algorithms can be *classified* according to whether they can handle *site failures and network partitioning* or *site failures only*. The *first* category of consistency control algorithms is those that can handle both site failures and network partitioning. Such algorithms, typically use a majority consensus (voting) approach to determine the majority partition. Several variants of voting algorithms have been proposed (see for example, [13, 23, 14, 15]). Voting based algorithms enable each partition to decide, *autonomously*, whether it contains a majority (K) of the total copies (N) of a data object, based on local state information maintained with each data object. Update and query operations can be performed within a majority partition. For more detailed discussion on this and other related issues see [22, 1].

The *second* category of consistency control algorithms is those that can handle site failures only. Two representative algorithms that belong to this category are ROWAA (Read-One-Write-All-Available) [6] and Available Copies[5]. These algorithms function by reading *any* available copy of the data object (preferably local) and writing into all available copies. Under these algorithms a user query can be always satisfied as long as at least *one* copy of the desired data object is available. Hence, our concern in this case is: at which sites of the network should copies of a given file be located to satisfy a specified minimum probability of the file being *available* when requested by users? This availability problem and our proposed solution are discussed in Section 2.2.

2.1 Quorum Reliability

In any replicated data management scheme involving majority consensus or voting type of consistency algorithms each copy of the data object carries one vote, or more generally an integer number of votes. Each read and write operation requires the collection of a majority of the votes assigned to a data object and this majority can be obtained in at most one partition called the *majority partition*. In order to determine the majority partition, a voting process has to be performed by the initiator of the read or update request. The quorum reliability of a system can therefore, be viewed as the product of:

- The probability of K or more of the N nodes being operational. This is referred to as the *node reliability*.

Distributed File Allocation with Consistency Constraints

Raj Tewari

Computer & Information Sciences
Temple University
Philadelphia, PA 19122

Nabil R. Adam

Graduate School of Management
Rutgers University
Newark, NJ 07102

Abstract

We consider the resource allocation problem in distributed computing systems that have strict mutual consistency requirements. Our model incorporates the behavior of consistency control algorithms, which ensure that mutual consistency of replicated data is preserved even when communication links of the computer network and/or computers on which the files reside fail. The problem of resource allocation in these networks is significant in terms of the efficiency of operations and the reliability of the network.

The constrained resource allocation problem is formulated as a mixed nonlinear integer program. An efficient algorithm is proposed to solve this problem. The performance of the algorithm is evaluated in terms of the algorithm's accuracy, efficiency and execution times, using a representative problem set.

1 Introduction

Consider a distributed computing system (DCS) that is made up of a set of sites (nodes) connected through communication links which transmit information from one site to another. Each site has a computer that processes user requests for some *common* data files.

If only one copy of that file is made available at a given site i , all related user requests initiated at that site would be processed locally. However, all related user requests initiated at the rest of the sites have to be transmitted to site i for processing and the results have to be transmitted back to the home sites. This one copy solution suffers from such drawbacks as: high communication costs incurred by the various sites, and low availability and reliability.

Another possible solution is to duplicate the common data file at each site in the network. In this case, higher availability and reliability would be achieved

and user queries at all sites would be processed locally thus, incurring no communication costs. On the other hand, this results in higher storage costs. In addition, in order to insure *mutual consistency* of replicated data, user update requests initiated at a given site have to be transmitted to all sites. Mutual consistency refers to having the same value for all copies of each data object at the various sites.

The file allocation problem was originally investigated by Chu [10]. He considered the problem: Given a number of computers that process common files, where should the files be located so that the minimum overall operating costs could be achieved. Chu's model was formulated as a nonlinear zero-one programming problem. By adding additional constraints, the problem was reduced to an equivalent linear zero-one programming problem.

Casey [9] studied the problem of allocating a given file in a computer network. The problem formulation used a linear integer program, and a solution procedure was developed using the hypercube method of enumeration. Casey applied his method to a five node network and a nineteen node ARPA network problems. Subsequent researchers have used Casey's problems for benchmark purposes.

Eswaran [11] proved that the *file allocation problem* (FAP) is NP-complete. This result motivated a number of heuristics proposed in the literature. Morgan and Levin [17] devised a heuristic for the allocation of program and data files in a computer network. This heuristic took into account the dependencies between data files and program files. Laning and Leonard [16] considered the case of a store-and-forward computer network. Their objective function was a sum of the storage costs and the message transmission costs, and the constraints included availability and performance constraints. This paper made an important contribution by introducing network performance constraints. The FAP with availability and performance constraints has been termed as *the general file allocation*