A Heuristic Task Assignment Algorithm to Maximize Reliability of a Distributed System

Gwo-Jen Hwang, Member IEEE
National Chiao Tung University, Hsin-Chu
Shian-Shyong Tseng, Member IEEE
National Chiao Tung University, Hsin-Chu

Key Words — **Distributed computer system, system reliability, task assignment, distributed software design**

Reader Aids —
General purpose: Propose an algorithm
Special math needed for explanations: Probability
Special math needed to use results: Same
Results useful to: Distributed-system designers

Summary & Conclusions — **Distributed systems potentially provide high reliability owing to the program and data-file redundancy possible. In many applications, high reliability is the major consideration for system design. Some work by Kumar, Hariri, Raghavendra shows that the distribution of programs and data-files can affect the system reliability appreciably, and that redundancy in resources such as computers, programs, and data-files can improve the reliability of distributed system. This paper first formulates a practical application for a reliability-oriented distributed task assignment problem which is NP-hard. Then, to cope with this challenging problem, we propose a *greedy* algorithm, based upon some heuristics, to find an approximate solution. The simulation shows that, in most cases tested, the algorithm finds suboptimal solutions efficiently; therefore, it is a desirable approach to solve these problems.**

1. INTRODUCTION

Distributed systems can provide appreciable advantages, including high performance, high reliability, resource sharing, and extensibility [5,17,19]. Potential reliability improvement of a distributed system is due to possible program & data-file redundancy. The reliability evaluation of distributed systems is widely published [1-4,6,11-13,15-16,20]. To evaluate the reliability of a distributed system, including a given distribution of programs and data-files, it is important to obtain a global reliability measure that describes how reliable the system is.

Kumar, Hariri, Raghavendra [9,13] proposed the concepts of distributed system reliability (DSR) which measures the reliability of a distributed system by determining the probability that all the distributed programs are working. They introduced the KHR[1] algorithm [9,13] based on graph theory to evaluate the reliability measures. Some of their previous work also shows that the distribution of programs and data-files can affect the system reliability appreciably [13-15], and that redundancy in resources such as computers, programs, and data-files can improve the reliability of distributed systems [8]. Therefore, the study of program and data-file assignment with consideration of redundancy is important in improving the DSR.

Since the evaluation of task's reliability is NP-complete [3], Hariri & Raghavendra [7] proposed an algorithm to solve some reliability-oriented task-allocation problems — assuming that each computer had the same reliability and each communication link had the same reliability [7]. Shatz & Wang solved the task allocation problems in redundant distributed-computer systems by assuming that the system is a cycle-free network [18]. In their model, software redundancy was not considered, and the mission reliability (continuous time interval that is sufficiently long for unit failures) was the major concern.

This paper formulates a practical application, of reliability-oriented design for a distributed information system, to the k-DTA problem; the k-DTA models the assignment of k copies of both distributed programs and their data-files to maximize the DSR under some resource constraints. Since the k-DTA problem is NP-hard [21], we then propose a greedy algorithm based upon some heuristics [10] to find an approximate solution. We conclude from the simulation results that in almost every case the approximate solution is suboptimal with relative error < 0.05 and the average absolute error $= 0.02$.

2. MOTIVATION OF THIS RESEARCH

The k-DTA problem originated from a project of installing several copies of a file server into the network which connects all of the universities in Taiwan ROC. The file server consists of a set of programs and a large database. The main purpose of having several copies of the server working at the same time is to ensure that the information system is not affected by local failures of computer sites or network links. If a server program can not access its data- files owing to disk failure, it can access the data of other operational copies through network to continue its work. If a host which holds some server programs fails, its users can still get the same services from other copies of the server.

There are many computers over several universities in question. Those schools are willing to offer their computer resources but with several resource constraints (*eg*, CPU time, memory space, disk quota). Therefore, the programs and data-files of each server are distributed among several computers under resource constraints on each computer. The schools are requested to list what their computers can afford (*eg*, how many processes, how many MB of memory space, how many GB of disk space), so that the whole information system can be planned.

[1] *Editors' note*: We have assigned this acronym for easy, clear reference.

Because a server is constantly executing, its reliability strongly depends on the failure probabilities of the associated computer sites and communication links. Therefore, the reliability of each computer or communication link can be evaluated according to the ratio of time periods of its historical failures. Section 4 puts this application into a formal description, in which a server is considered as a distributed system consisting of several programs and data- files; each communication link is a link in a graph with a reliability measure, and each computer is a node in the graph with a reliability measure and some resource constraints.

3. NOTATION & DEFINITIONS

Bidirectional communication channels operate between processing elements. A distributed network can be modeled by a simple undirected graph.

Notation & Acronyms

AR-tree access-relation tree
file data-file
G simple undirected graph: (V,E)
V set of nodes representing the processing elements
E set of links representing bidirectional communication channels
DSR distributed-system reliability
KHR Kumar, Hariri, Raghavendra reliability-computing algorithm [9,13]
X_i node representing processing element i
L_{ij} link between X_i *and* X_j
$R(X_i)$ Pr{X_i is operational}
$R(L_{ij})$ Pr{L_{ij} is operational}
P_i distributed program i
F_i file i
PF_i distributed program or file i
AFL(P_i) list of files required for program i to complete its execution
APL(F_i) list of programs which must access file i to complete their executions
FST file spanning tree consisting of the root node (processing element that runs the program) and some other nodes which hold all the files needed for the program held in the root node under consideration [13,15]
MFST minimal FST containing no subset file spanning tree [15]
MFST(P_i) set of minimal file spanning trees associated with program i
ASS(S,G) assignment which allocates all programs & files to a set of nodes S of network G
DSR(S,G) DSR for ASS(S,G)
FSF file spanning forest: a set of FSTs whose root nodes hold all of the programs under consideration [15]
MFSF minimal FSF containing no subset FSF
DTA distributed task assignment
k-DTA k-copies DTA.

Other, standard notation is given in "Information for Readers & Authors" at the rear of each issue.

Definitions

3.1 Distributed system: A system involving cooperation among several loosely coupled computers (processing elements); the system communicates (by links) over a network.

3.2 Distributed program: A program of some distributed system which requires one or more files. For successful execution of a distributed program, the local host, the processing elements having the required files, and the interconnecting links must all be operational [13,15].

3.3 DSR: Pr{all the specified distributed programs for the system are operational} [13,15].

3.4 Dependent set: A set S of distributed programs & files such that there does not exist a partition which divides S into two disjoint subsets S_1 & S_2, where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \emptyset$ such that each program and the files required are within the same subset.

3.5 DTA problem: Find an assignment for a dependent set under some resource constraints on the distributed system such that the distributed system reliability is maximum.

3.6 k-DTA problem: Find an assignment for k copies of a dependent set to maximize the DSR under some resource constraints on the distributed system. ◄

Example 3.1

Let AFL(P_1) = {F_1, F_2}, AFL(P_2) = {F_2, F_3}.

According to definition 3.4, S . {P_1, P_2, F_1, F_2, F_3} is a dependent set. If P_2 requires only file F_3, S is not a dependent set since S can be divided into S_1 = {P_1, F_1, F_2} and S_2 = {P_2, F_3} such that both S_1 & S_2 are dependent sets. ◄

Assume that in a dependent set, one arbitrary program is not operational or can not access the required file because of the failure of network nodes or links. Then all other programs of the dependent set must stop executing. In example 3.1, let P_1 & P_2 be 2 processes of a parallel algorithm, it is pointless for P_2 to continue executing if P_1 has already halted due to failure of some nodes or links. By definition, the operation of a dependent set S relies on the operation of the programs & files of S. Therefore, the reliability of a dependent set in the distributed system can be evaluated by KHR which measures the DSR by determining the probability that all the distributed programs are working.

This paper is concerned with the assignments of dependent sets to maximize the distributed system reliability evaluated by KHR.

4. HEURISTIC ALGORITHM FOR THE k-DTA ProbLEM

4.1 Background

This section proposes an efficient heuristic algorithm to find an approximate solution of the k-DTA problem. Without loss of generality, we use *memory* constraints instead of *resource* constraints to simplify the discussion.

Theorem 1. Denote the set of minimal file spanning trees for an assignment ASS(S,G) of a dependent set by MFST(S,G). If there exists another assignment ASS$(S-\{v\},G)$, where v is a terminal node of some MFST in MFST(S,G), then DSR(S,G) < DSR$(S-\{v\},G)$.

Proof: The theorem is obvious from the definition of DSR. ◄

Definitions

4.1 Node X_1 is *more reliable* than node X_2 iff the degree of X_1 is higher than that of X_2. [The node with higher degree is more likely to have more paths to the destination nodes than those with lower degrees. Thus according to DSR$(S,G) = \cup_i$ Pr{MFSF$_i$}, X_1 could provide higher reliability than X_2.]

4.2 Program P_1 is *weaker* than program P_2 (or, P_2 is *stronger* than P_1) iff the minimum number of nodes required to assign P_1 and its associated files are greater than those required for P_2. [If any associated file is not accessible, the program fails. From theorem 1, we can always find an assignment such that P_2 is in a more reliable situation than P_1; therefore, P_1 is weaker than P_2.]

4.3 File F_1 is more *influential* than file F_2 iff F_1 is accessed by more programs than F_2. [By definition of DSR, if any program can not access its associated file, the whole distributed system fails. Therefore, if F_1 is accessed by more programs than F_2, then the probability that some program can not access F_1 is likely to be greater than the probability for F_2.]

Finding the minimum number of nodes needed to hold a program and its associated files is interesting and difficult. Basically, in most cases the total required memory size dominates the number of nodes needed; therefore, we simply use the total memory size of P_i and its associated files to approximate the number of nodes required. The weakness decision function is:

$$\text{WEAKNESS}(P_i) \equiv \text{SIZE}(P_i) + \Sigma_{F_j \in \text{AFL}(P_i)} \text{SIZE}(F_j).$$

Example 4.1

All the program & files are the same size. The access relations of a dependent set are:

$$\text{AFL}(P_1) = \{F_1, F_2, F_3\}$$

$$\text{AFL}(P_2) = \{F_2, F_3, F_4, F_5\}$$

$$\text{AFL}(P_3) = \{F_1, F_3\}$$

$$\text{AFL}(P_4) = \{F_2, F_3\}.$$

The order of the programs from weakest to strongest is:

$$P_2, P_1, P_3, P_4.$$

Theorem 2. The most reliable assignment for k copies of some program or file is to assign these copies to k distinct nodes.

Proof: The theorem is obvious from the definition of DSR. ◄

Heuristics 1 - 6 are ideas about approximate efficient solution of the task assignment problem.

Heuristics

1. Assign the weaker programs first.
2. Assign the weaker programs to the more reliable nodes.
3. Assign the more influential files before the less influential ones.
4. Assign the more influential files to the more reliable nodes.
5. Assign the copies of the same program or file to different nodes.
6. Assign a program as close to its files as feasible.

4.2 Access Relation Tree

Our approximation algorithm is generally a greedy approach which uses heuristics 1 - 6 as optimization measures and an AR-tree as the data structure to represent the requirement relation between the distributed programs and their files. To construct an AR-tree —

a. Assign the weakest program, say P_i, to the root of the AR-tree.

b. Assign each file in AFL(P_i) to the P_i-children nodes.

c. Assign the programs in APL(F_j) to the F_j-children nodes, *etc.*

That is, AFL(P_i) & APL(F_j) are assigned alternately to the children nodes of P_i & F_j for each P_i on odd depth and F_j on even depth of the AR-tree. Moreover, the children of the same parent are assigned sequentially from *weakest* to *strongest*.

Construction of AR-tree for Example 4.1

1. Determine the order of the programs from weakest to strongest (see example 4.1).

2. Construct the APL(F_i) for each F_i and determine the order of these files from most influential to least influential:

$$\text{APL}(F_1) = \{P_1, P_3\}$$

$$\text{APL}(F_2) = \{P_1, P_2, P_4\}$$

$$\text{APL}(F_3) = \{P_1, P_2, P_3, P_4\}$$

$$\text{APL}(F_4) = \{P_2\}$$

$$\text{APL}(F_5) = \{P_2\}$$

According to the size of APL(F_i), the order of the files is F_3, F_2, F_1, F_4, F_5.

3. Sort the elements of AFL(P_i) & *APL*(F_j) for each P_i and each F_j according to the orders determined in steps 1 & 2:

$$\text{AFL}(P_1) = \{F_3, F_2, F_1\}$$

$AFL(P_2) = \{F_3, F_2, F_4, F_5\}$

$AFL(P_3) = \{F_3, F_1\}$

$AFL(P_4) = \{F_3, F_2\}$.

$APL(F_1) = \{P_1, P_3\}$

$APL(F_2) = \{P_2, P_1, P_4\}$

$APL(F_3) = \{P_2, P_1, P_3, P_4\}$

$APL(F_4) = \{P_2\}$

$APL(F_5) = \{P_2\}$

4. Assign each program and file alternately to form an AR-tree. Initially, assign the weakest program P_2 to the root of the AR-tree. Assume $k=2$ (2 copies of each program & file). The following rules are used to assign the programs & files until all of them appear twice:

A. $AFL(P_i)$ is assigned to the children nodes of P_i. If any file appears more than k (in this example, $k=2$) times, discard it.

B. If $F_j \in AFLP_i)$ and F_j has been assigned to the parent node of Pi, reorder F_j to be the last (rightmost) child of P_i to be assigned.

C. Assign $APL(F_i)$ to the children nodes of F_i. If any program appears more than k times, discard it.

D. If $P_j \in APL(F_i)$ and P_j has been assigned to the parent-node of F_i, then reorder P_j to be the last (rightmost) child of F_i to be assigned.

E. If all modules appear k times, STOP the extension of the AR-tree; else go back to A.

After several iterations, an AR-tree of example 4.1 is constructed as figure 1. ◄

The AR-tree represents two important relations among the k copies of programs & files in a dependent set:

1. Parent & children have access relation; hence they should be put as near as possible.

2. For the programs or files of the same parent, the left one is weaker than the right one; therefore, the priority of assignment should decrease from left to right. ◄

For these reasons, it seems that better solutions can result from assigning the programs & files in breadth-first order.

4.3 Greedy Approach

After the AR-tree is constructed, the greedy algorithm based upon some heuristics is used to assign the programs & files to the network:

1. Initially the program in the root of AR-tree is selected and assigned to the node with the maximum *environment weight*. The environment weight represents the composite reliability for the nodes and links surrounding a node.

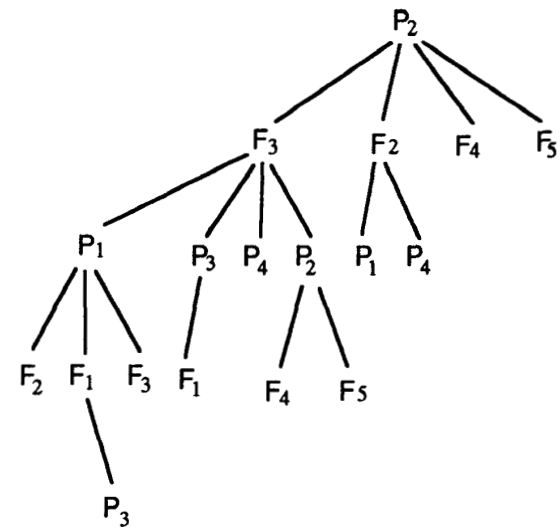

Figure 1. An AR-Tree of Example 4.1

$$\text{ENVIRONMENT\_WEIGHT}(X_i) \equiv R(X_i) \cdot \Sigma_{X_j \in \text{ADJ}(X_i)} R(X_j) \cdot R(L_{ij}), \quad (4\text{-}1)$$

$ADJ(X_i)$ = set of nodes which are adjacent to X_i.

2. Once a program or file PF_t is assigned to some node X_j, we then try to assign the children of PF_t to the nodes as close to X_j as possible. The children of PF_t are assigned from weakest to strongest.

3. For some child of PF_t, say PF_k, we try first to assign PF_k to X_j if the current available memory of X_j is large enough to hold PF_k; otherwise, the *i-movement nodes* of X_j (for such nodes, all of the paths to X_j must include at least i edges and at least 1 path including exactly i edges) are tried for $i = 1,2,\ldots,n-1$. In figure 2, the 1-movement nodes of X_3 are: X_1, X_2, X_5; the 2-movement nodes of X_3 are: X_4, X_6.

4. If there are at least 2 i-movement nodes with enough memory space, assign PF_k to the one with maximum *access weight*. As an i-movement node, X_S becomes the candidate node, the access weight for X_S is:

$$\text{ACCESS\_WEIGHT}(X_j, X_S) \equiv \text{MAX}_{PA}\left\{\left(\prod_{L_{uv} \in PA} R(L_{uv})\right) \cdot \left(\prod_{X_f \in PA} R(X_f)\right)\right\},$$

PA = some i-movement path from X_j to X_S.

Access weight is an estimate of the reliability of assigning PF_k to some i-movement node X_S while the parent of PF_t is in X_j; it considers the degree of parent node and the reliabilities of

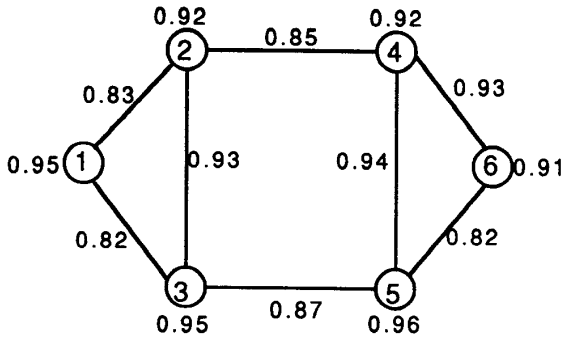

Figure 2. Example of Network Topology

associated nodes & links. For each pair of nodes, the access weight can be found by the *all-pairs-shortest-paths* algorithm [10].

5. If the breadth-first order of PF_{k_2} is greater than that of PF_{k_1} in the AR-tree, then PF_{k_1} must be assigned before PF_{k_2}. This rule assures that the programs & files in AR-tree are assigned to the network according to the priority of breadth-first order. ◀

For programs of short execution time, the process reliability is related to its execution time, communication time, and current system load; hence the environment weights and access weights will be changed. However, in considering the constant (long term) execution, the program reliability strongly depends on the failure probabilities of relative nodes & links; *ie*, the environment weights and access weights are constants in such case.

Example 4.2

For a dependent set $\{P_1, F_1, F_2\}$ to be assigned to the network of figure 2, let the available memory space of each node be $C_1 = 8$, $C_2 = 8$, $C_3 = 12$, $C_4 = 8$, $C_5 = 10$, $C_6 = 10$. Let $\text{AFL}(P_1) = \{F_1, F_2\}$, $\text{SIZE}(P_1) = 4$, $\text{SIZE}(F_1) = 6$, $\text{SIZE}(F_2) = 7$, and the number of copies $k = 2$.

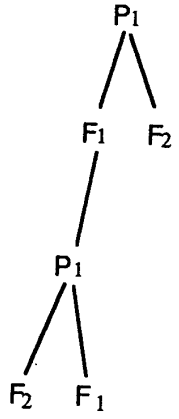

Figure 3. An AR-Tree of Example 4.2

The AR-tree in figure 3 can be constructed. First compute the environment weight for each node:

ENVIRONMENT_WEIGHT(1) = 1.465470

ENVIRONMENT_WEIGHT(2) = 2.257680

ENVIRONMENT_WEIGHT(3) = 2.346310

ENVIRONMENT_WEIGHT(4) = 2.328244

ENVIRONMENT_WEIGHT(5) = 2.340000

ENVIRONMENT_WEIGHT(6) = 1.494948.

According to the environment weights and the AR-tree, initially P_1 is assigned to X_3. The children of P_1 (*viz*, F_1, F_2) are the next two files to be assigned. F_1 is assigned to node X_3; hence C_3 becomes 12 - (4+6) = 2. Since $\text{SIZE}(F_2) = 7$, X_3 does not have enough available memory to hold F_2; therefor, the X_3 1-movement nodes (*viz*, X_1, X_2, X_5) which have enough memory space to hold F_2 are tried. Their access weights are:

$\text{ACCESS\_WEIGHT}(X_3, X_1) = 0.740050,$

$\text{ACCESS\_WEIGHT}(X_3, X_2) = 0.812820,$

$\text{ACCESS\_WEIGHT}(X_3, X_5) = 0.793440.$

Since X_2 has the maximum access weight associated with X_3, then F_2 is assigned to X_2. We then consider the second copy of P_1 in depth 3 of the AR-tree (we call it P'_1). Since P'_1 is the child of F_2, and since F_2 has been assigned to X_2, therefore, we try to assign P'_1 to X_2 first. However, the available memory space of X_2 is 8 - 7 = 1 < $\text{SIZE}(P'_1)$ = 4; so, the 1-movement nodes of X_2 are tried.

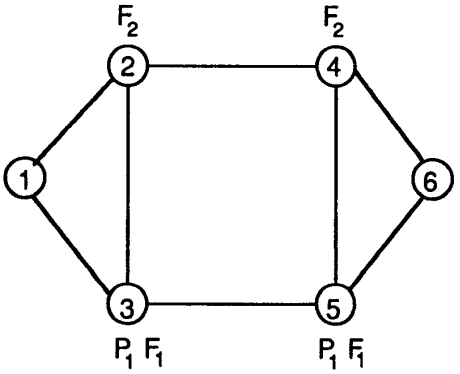

Figure 4. Result of Applying the Greedy Algorithm

The algorithm continues iterating until k copies of all the programs & files are assigned to the network. The final result of example 4.2 is in figure 4. By KHR, the distributed system reliability of figure 4 is 0.980363.

4.3 Formal Description of the Greedy Algorithm

Greedy Algorithm

1. Calculate the environment weight of each X_i.
   1.1 ADJ(X_i) = {X_j|X_j is adjacent to X_i}
   1.2 ENVIRONMENT_WEIGHTX_i) = [use (4-1)]
2. Construct an Assignment-Relation tree.
   2.1 For each P_i & F_j, reorder the AFL(P_i) & APL(F_j).
   2.2 Choose P_h as the root of the AR-tree, WEAKNESS(P_h) = MAX$_i$(SIZE(P_i) + $\Sigma_{F_j \in AFL(P_i)}$ SIZE(F_j)).
   2.3 /* $D \equiv$ depth of the current AR-tree) */ $D=0$
   UNTIL k copies of all programs & files are in the AR-tree DO
   2.3.1 IF D is odd
   THEN expand the tree for eachP_i of level D; the files of AFL(P_i) become the children of P_i
   ELSE expand the tree for each F_i of level D; the programs of APL(F_i) become the children of F_i
   END_IF
   2.3.2 For the new expanded programs or files, remove the excess ones if more than k copies are being expanded
   2.3.3 IF (PF$_j$ is expanded from PF$_i$) AND (PF$_j$ = parent of PF$_i$)
   THEN reorder PF$_j$ to be the rightmost child of PF$_i$
   END_IF
   2.3.4 D = D + 1
   END_UNTIL
3. Apply the all-pairs-shortest-paths algorithm to find the access weight for each pair of nodes in the distributed system.
4. Assign each program & file to the network according to the greedy strategy and the AR-tree.
   4.1 Choose node X_S which has the maximum environment weight. Assign the root program of AR-tree to X_S.
   4.2 Select one program or file PF$_j$ from the AR-tree according to the breadth-first order
   UNTIL all programs & files are selected DO
   4.2.1 Assume that PARENT(PF$_j$) is assigned to X_c
   4.2.2 $i = 0$
   4.2.3 WHILE none of the i-movement nodes of X_c has enough resources for PF$_j$ DO
   $i = i+1$
   END_WHILE
   4.2.4 Choose 1 node X_r from the i-movement nodes of X_c that have maximum access weight
   4.2.5 Assign PF$_j$ to X_r and update the available resources of X_r
   END_UNTIL ◄

Notation

M	$k\cdot$(number of programs & files)
N	number of nodes in the distributed system.

The time complexities for the algorithm are:

Step	Complexity
1	$O(N)$
2	$O(M)$
3	$O(N^3)$
4	$O(M\cdot N)$
Total	$O(N^3 + M\cdot N)$.

5. SIMULATION & ANALYSIS

To evaluate the performance of our approach, we applied the heuristic algorithm to a wide variety of distributed task assignment problems.

5.1 Quantitative Evaluation

The results of our algorithm are compared to those of a *Random Assignment* algorithm and an *Exhaustive Search* algorithm. The reliabilities of the assignments were evaluated by applying KHR. To verify the accuracy & efficiency of our algorithm, the simulation programs are implemented in C language on a VAX-8800 and by COMMON LISP on a PC/AT, respectively. Some parts of the simulation results are depicted in tables 1 & 2. Since k (number of copies) = 2, the numbers of programs & files are twice as large as those shown in the tables. Two error measures, E_r & E_{ave}, are used.

Notation

E_r	relative error
E_{ave}	average absolute error
DSR_{app}	solution of the approximation algorithms
$DSR_{optimal}$	optimal solution from exhaustive-search algorithm.

$E_r \equiv 1 - DSR_{app}/DSR_{optimal}$

$E_{ave} \equiv (\Sigma|DSR_{optimal} - DSR_{app}|)/$(number of cases)

The simulation shows that our algorithm performs accurately & efficiently for most cases without dependence on the languages or computers used:

$E_r < 0.05$ for each case

$E_{ave} = 0.02$ on the average.

Therefore, in almost every case, our algorithm can find suboptimal assignments.

TABLE 1
Simulation Results of C Program Executed on VAX-8800

Size			Random Alg.		Greedy Alg.			Exhaustive Search	
L	P	F	DSR	E_r	time (sec)	DSR	E_r	time (min)	DSR
8	1	2	0.924	0.068	0.07	0.971	0.020	18.4	0.991
8	1	2	0.925	0.067	0.08	0.971	0.020	18.6	0.991
8	1	2	0.902	0.088	0.08	0.980	0.009	18.6	0.989
8	1	2	0.933	0.058	0.07	0.982	0.009	18.4	0.991
8	1	2	0.903	0.080	0.08	0.972	0.010	19.1	0.982
8	1	2	0.824	0.157	0.06	0.931	0.048	10.7	0.97
8	1	2	0.871	0.112	0.07	0.956	0.026	18.7	0.98
8	1	2	0.852	0.122	0.08	0.951	0.021	19.2	0.97
8	1	2	0.897	0.093	0.07	0.968	0.022	18.8	0.99
8	1	2	0.860	0.123	0.06	0.965	0.016	19.1	0.98
8	1	2	0.844	0.131	0.07	0.951	0.021	18.6	0.97
9	1	1	0.909	0.082	0.06	0.989	0.001	48.2	0.990
9	1	1	0.915	0.077	0.05	0.975	0.017	46.6	0.992
9	1	1	0.918	0.073	0.06	0.989	0.002	49.1	0.990
9	1	1	0.785	0.185	0.06	0.944	0.020	48.5	0.963
9	1	1	0.781	0.182	0.05	0.947	0.008	48.4	0.955
8	2	2	0.895	0.094	0.06	0.961	0.027	934	0.988
8	1	3	0.908	0.081	0.07	0.974	0.015	928	0.989
8	2	3	0.907	0.069	0.09	0.960	0.014	$>10^4$	0.974
8	2	4	0.911	—	0.09	0.954	—	$>10^6$	—
10	1	2	0.932	0.039	0.07	0.967	0.003	616	0.970
10	1	2	0.886	0.083	0.08	0.950	0.016	597	0.965
10	1	3	0.918	0.054	0.10	0.968	0.001	$>10^4$	0.970
10	2	3	0.946	—	0.12	0.973	—	$>10^5$	—

E_{ave}(Random) = 0.094; E_{ave}(Greedy) = 0.015
L = number of Links
P = number of programs
F = number of data files
1 year $\approx 5 \cdot 10^5$ minutes; 1 month $\approx 4 \cdot 10^4$ minutes

TABLE 2
Simulation Results of LISP Program Executed on PC/AT

Size			Random Alg.		Greedy Alg.			Exhaustive Search	
L	P	F	DSR	E_r	time (sec)	DSR	E_r	time (min)	DSR
8	1	2	0.924	0.068	0.16	0.971	0.020	31.9	0.991
8	1	2	0.925	0.067	0.17	0.971	0.020	37.4	0.991
8	1	2	0.902	0.088	0.17	0.980	0.009	37.5	0.989
8	1	2	0.933	0.058	0.16	0.982	0.009	37.5	0.991
8	1	2	0.903	0.080	0.17	0.972	0.010	37.7	0.982
8	1	2	0.824	0.157	0.15	0.931	0.048	21.2	0.978
8	1	2	0.871	0.112	0.16	0.956	0.026	37.5	0.982
8	1	2	0.852	0.122	0.17	0.951	0.021	37.4	0.971
8	1	2	0.897	0.093	0.16	0.968	0.022	37.5	0.990
8	1	2	0.860	0.123	0.16	0.965	0.016	37.7	0.981
8	1	2	0.844	0.131	0.16	0.951	0.021	37.8	0.971
9	1	1	0.909	0.082	0.16	0.989	0.001	92.3	0.990
9	1	1	0.915	0.077	0.11	0.975	0.017	95.8	0.992
9	1	1	0.918	0.073	0.16	0.989	0.002	92.4	0.990
9	1	1	0.785	0.185	0.16	0.944	0.020	97.7	0.963
9	1	1	0.781	0.182	0.16	0.947	0.008	92.9	0.955
8	2	2	0.895	0.094	0.16	0.961	0.027	1770	0.988
8	1	3	0.908	0.081	0.17	0.974	0.015	1696	0.989
8	2	3	0.907	0.069	0.18	0.960	0.014	$>10^5$	0.974
8	2	4	0.911	—	0.19	0.954	—	$>10^6$	—
10	1	2	0.932	0.039	0.16	0.967	0.003	3100	0.970
10	1	2	0.886	0.083	0.17	0.950	0.016	1013	0.965
10	1	3	0.918	0.054	0.22	0.968	0.001	$\approx 10^5$	0.970
10	2	3	0.946	—	0.28	0.973	—	$>10^6$	—

E_{ave}(random) = 0.094; E_{ave}(greedy) = 0.015
[see footnotes on table 1]

5.2 *Qualitative Evaluation*

We analyze the performance of our algorithm by comparing it with the same *Random Assignment* algorithm used in section 5.1 and with the heuristic Algorithm-*S*.

Algorithm-S

1. Assign programs to the most reliable and allowable (see #3) node which has enough memory.
2. Assign each file to the allowable (see #3) node which holds the most programs that access it.
3. If the node has held a copy of some program or file, do not assign the same one to it. ◄

Discussion of Algorithm-S (AlgS)

AlgS seems straight-forward and reasonable; however, AlgS has 2 problems.

Problem #1: AlgS ignores the relationships among copies of the modules. For example, let P_i & P'_i be 2 copies of the same program, and similarly with F_i & F'_i. If AFL(P_1) = $\{F_1, F_2\}$; then AlgS implies that only P_1 or P'_1 needs to access one of $\{F_1, F_2\}$, $\{F'_1, F_2\}$, $\{F_1, F'_2\}$, $\{F'_1, F'_2\}$ to keep the whole system operational.

Problem #2: AlgS does not decide the order of assigning modules by considering their relationships; hence a program is most likely to be assigned far from the files it needs, and vice versa. For example, the assigning orders of F_1 & F_2 can be much later than that of P_1. Let P_1 be assigned to X_c. The nodes that are close to X_c might already be occupied when F_1 & F_2 are going to be assigned. ◄

Discussion of Random-Assignment Algorithm (AlgR)

AlgR works in an even simpler way: AlgR applies only rule #3 of Algorithm-S shown above.

Problem #3: The performance of AlgR can be terrible if most of the modules are assigned to some unreliable nodes which are far apart. ◄

Problems #1 & #2 for Algorithm-S can be avoided by applying the AR-tree.

Problem #3 for Algorithm-R usually can be solved by applying environment weights and access weights which lead the algorithm to assign the modules to more reliable nodes with more reliable links surrounded. Therefore, satisfiable results can be derived by our approach.

ACKNOWLEDGMENT

This research was partially supported by the National Science Council of the Republic of China under Contract NSC79-0408-E009-17.

REFERENCES

[1] K.K. Aggarwal, S. Rai, "Reliability evaluation in computer-communication networks", *IEEE Trans. Reliability*, vol R-30, 1981 Apr, pp 32-35.

[2] A. Agrawal, R.E. Barlow, "A survey of network reliability and domination", *Operations Research*, vol 32, 1984 May, pp 478-492.

[3] M.O. Ball, "Complexity of network reliability computation", *Networks*, vol 10, 1980, pp 153-165.

[4] D. Brown, "A computerized algorithm for determining the reliability of redundant configurations", *IEEE Trans. Reliability*, vol R-20, 1971 Aug, pp 121-124.

[5] P. Enslow, "What is a distributed data processing system", *Computer*, vol 11, 1978 Jan, pp 13-21.

[6] S. Hariri, C.S. Raghavendra, "SYREL: A symbolic reliability algorithm based on path and cutset methods", *IEEE Trans. Computers*, vol C-36, 1987 Oct, pp 1224-1232.

[7] S. Hariri, C.S. Raghavendra, "Distributed functions allocation for reliability and delay optimization", *Proc. IEEE/ACM 1986 Fall Joint Computer Conf.*, 1986, pp 344-352.

[8] S. Hariri, C.S. Raghavendra, V.K.P. Kumar, "Reliability analysis in distributed systems", *Proc. 6th Int'l Conf. Distributed Computing Systems*, 1986 May, pp 564-571.

[9] S. Hariri, C.S. Raghavendra, V.K.P. Kumar, "Reliability measures for distributed processing systems", *Proc. Int'l Symp. New Directions in Computers*, 1985 Aug; Trondheim, Norway.

[10] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms*, 1978; Computer Science.

[11] C.L. Hwang, F.A. Tillman, M.H. Lee, "System reliability evaluation techniques for complex large systems - A review", *IEEE Trans. Reliability*, vol R-30, 1981 Dec, pp 411-423.

[12] K.B. Misra, "An algorithm for the reliability evaluation of redundant networks", *IEEE Trans. Reliability*, vol R-19, 1970 Nov, pp 146-151.

[13] V.K.P. Kumar, C.S. Raghavendra, S. Hariri, "Distributed program reliability analysis", *IEEE Trans. Software Engineering*, vol SE-12, 1986 Jan, pp 42-50.

[14] C.S. Raghavendra, S. Hariri, "Reliability optimization in the design of distributed systems", *IEEE Trans. Software Engineering*, vol 11, 1985 Oct, pp 1184-1193.

[15] C.S. Raghavendra, V.K.P. Kumar, S. Hariri, "Reliability analysis in distributed systems", *IEEE Trans. Computers*, vol 37, 1988 Mar, pp 352-358.

[16] S. Rai, K.K. Aggarwal, "An efficient method for reliability evaluation", *IEEE Trans. Reliability*, vol R-27, 1978 Jun, pp 101-105.

[17] D.A. Rennel, "Distributed fault-tolerant computer systems", *Computer*, vol 13, 1980 Mar, pp 55-56.

[18] S.M. Shatz, J.P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant-computer systems", *IEEE Trans. Reliability*, vol 38, 1989 Apr, pp 16-27.

[19] J.A. Stankovic, "A perspective on distributed computer systems", *IEEE Trans. Computers*, vol C-33, 1984 Dec, pp 42-50.

[20] A. Satyanarayana, "A unified formula for analysis of some network reliability problems", *IEEE Trans. Reliability*, vol R-31, 1982 Apr, pp 23-32.

[21] S.S. Tseng, G.J. Hwang, "Task assignment to maximize reliability of a distributed system is NP-hard", *Tech. Report NCTU-CC-80123001*, 1991; National Chiao Tung University, Taiwan - ROC.

AUTHORS

Dr. Gwo-Jen Hwang; Computer Center; National Chiao Tung University; Hsinchu 300 TAIWAN - R.O.CHINA.

Gwo-Jen Hwang (M'92) was born 1963 April 16 in Taiwan - ROC. In 1991, he received his PhD from the Department of Computer Science and Information Engineering at National Chiao Tung University in Taiwan. He is now an Associate Professor and is Director of R&D Department of Computer Center at that university. His research interests are distributed systems, expert systems, and knowledge engineering.

Dr. Shian-Shyong Tseng; Department of Computer and Information Science; National Chiao Tung University; Hsinchu 300 TAIWAN - R.O.CHINA.

Shian-Shyong Tseng received his PhD in Computer Engineering from National Chiao Tung University in 1984. He is now a Professor in the Department of Computer and Information Science at National Chiao Tung University, and is Director of Computer Center at Ministry of Education. He was elected an Outstanding Talent of Information Science of R.O.C. in 1989 and was awarded the Outstanding Youth Honor of R.O.C. in 1992. His research interests include computer algorithms, distributed / parallel computing, artificial intelligence, and compilers.

Manuscript TR89-224 received 1989 December 29; revised 1991 January 10; revised 1992 January 16; revised 1992 September 4.

IEEE Log Number 06540 ◄TR►