JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 17, 633-646 (2001)

Short Paper

File Allocation Algorithms to Minimize Data Transmission Time in Distributed Computing Systems*+

PAO-YUAN CHANG, DENG-JYI CHEN** AND KRISHNA M. KAVI++
Information Management Department
Ta Hwa Institute of Technology
Hsinchu, Taiwan 307, R.O.C.
E-mail: pychang@et4.thit.edu.tw
**Computer Science and Information Engineering Department*
National Chiao Tung University,
Hsinchu, Taiwan 300, R.O.C.
E-mail: djchen@csie.nctu.edu.tw
++Department of Electrical and Computer Engineering*
University of Alabama in Huntsville
Huntsville, AL 35899, U.S.A.
E-mail: kavi@ebs330.eb.uah.edu

This work addresses a files allocation problem (FAP) in distributed computing systems. This FAP attempts to minimize the expected data transfer time for a specific program that must access several data files from non-perfect computer sites. We assume that communication capacity can be reserved; hence, the data transmission behavior is modeled as a many-to-one multi-commodity flow problem. A new critical-cut method is proposed to solve this reduced multi-commodity flow problem. Based on this method, two algorithms which use branch-and-bound are proposed for this FAP. The proposed algorithms are able to allocate data files having single copies or multiple replicated copies. Simulation results are presented to demonstrate the performance of the algorithms.

Keywords: distributed computing system (DCS), cut, multi-commodity flow, linear programming, data replication, branch and bound

1. INTRODUCTION

Due to recent advances in microprocessor technologies, distributed computing systems (DCS) have been increasingly used in diverse fields involved in time critical applications, such as industrial manufacturing, traffic control, and the military. Successful

Received December 3, 1998; revised May 29, 1999; accepted September 7, 1999.
Communicated by Arbee L. P. Chen.
* This research work is supported in part by National Science Council in Taiwan under Contract No. NSC872213E009094.
+ The previous version of this paper was published in Proceedings of IEEE International Computer Performance and Dependability Symposium (IPDS), Sep. 4-6, 96, Urbanna-Champaign, Illinois, U.S.A.

execution of an application hinges on not only successful operation of computing devices, but also on the program response time. In a DCS, network efficiency plays a prominent role in determining the program response time. The running program and the data files are often placed at different computer sites; thus, execution requires remote file access. To shorten the data transfer time, the allocation of data files such that the data transfer time can be minimized becomes an important design issue. This kind of problem dealing with the assignment of files to processing nodes so as to optimize performance is commonly known as a file allocation problem (FAP). Many types of FAPs with different objectives and constraints have been addressed in the literature [2-4, 6, 7, 9-14]. Typically, FAPs are complex integer programming problems with no known efficient solutions [5]. Hence, heuristic solutions are needed.

In this paper, we consider the FAP in a DCS based on packet-switched subnet. The primary goal of this FAP is to minimize the data transfer time for a specific program which needs data from several data files in order to execute. We assume that each link has reserved a fixed communication capacity for this specific program. Based on the fact that the size of a packet is much less than that of a data file, a flow model can accurately reflect the data transmission behavior. In the flow model, a DCS is presented by an undirected graph, where nodes and links denote computer sites and communication links, respectively, and the numbers associated with links denote the reserved link capacities. The nodes which hold data files are sources, and the node where the program is located is the target. Data transmission is, thus, modeled as a many-to-one multi-commodity flow problem. A linear programming technique is, therefore, a feasible way to solve this problem.

Besides minimizing the data transfer time, the FAP also attempts to maximize program reliability. However, it is not always possible to find an allocation that can optimize both performance measures. When this dilemma occurs, minimization of the data transfer time will be the considered first. This strategy is based on the following facts. Firstly, current computer facilities are highly reliable. A small improvement in reliability, for example 10^{-6}, is meaningless for most users. Secondly, program reliability can be easily improved through the use of redundancy in data files. That is, files can be replicated and placed at different computer sites safeguard against the situation in which some of the computer sites fail.

This paper presents two algorithms, NOFA (Near Optimal File Allocation) and HFA (Heuristic File Allocation), to deal with the FAP that has the objectives stated above. These algorithms are based on branch-and-bound, and are only different in the number of steps they traverse in the search tree. NOFA traverses more steps than HFA does and, thus, produces better results. Both algorithms are capable of allocating replicated copies, but neither can guarantee that the optimal allocation will be found. Herein, the optimal allocation means that the expected data transfer time from the sources to the target is the minimum (given that program execution is successful). NOFA is considered to be "near optimal" since for highly reliable nodes, likelihood that NOFA will find the optimal allocation is very high. If the nodes are perfect, NOFA will always find the optimal allocation.

2. PROBLEM DEFINITION

Let m be the number of data files to be allocated, and let n be the number of computer sites (i.e. the nodes in the undirected graph) in the DCS. We label the data files and computer sites $F_1, F_2, \ldots, F_m$ and $N_1, N_2, \ldots, N_n$, respectively. Each data file may be replicated and allocated on different nodes so that one nodal failure the program execution. Hence, an allocation matrix $M = [M_{ij}]$ of the data files is defined as an $m \times n$ matrix, where

$$M_{ij} = \begin{cases} 0 & \text{if } F_i \text{ is not allocated on } N_j \\ k\ (>0) & \text{if the } k\text{th copy of } F_i \text{ is allocated on } N_j \end{cases} .$$

Program P firstly attempts to read data files of primary copies (i.e., the first copies). If some of them are not available, their second copies will be retrieved instead. Therefore, the kth copy of a data file is read by the program P only when accessing the k-1 copies (from the first to the (n-1)th copies) fails. Therefore, we say that an allocation matrix is valid if, for every i, $\underset{j=1}{\overset{n}{MAX}}(M_{ij})$ is greater than zero, and if there is exact one element in the ith row whose value is k', for k' = 1, 2, ..., $\underset{j=1}{\overset{n}{MAX}}(M_{ij})$.

The cost function is defined as the expected value of the data transfer time. Let an *allocation vector A* be an m-element row vector which indicates from which nodes data files are transferred. For a given valid allocation Matrix M, a valid allocation vector is one of the following: $(S(F_1^{k1}), S(F_2^{k2}), \ldots, S(F_m^{km}))$ for $1 \le ki \le \underset{j=1}{\overset{n}{MAX}}(M_{ij}), 1 \le i \le m$, where F_i^k denotes the kth copy of F_i and $S(F_i^k)$ is a function returning the node number on which F_i^k is allocated, i.e., $S(F_i^k) = j$ if $M_{ij} = k$. The cost function of computing the expected data transfer time is, thus, as follows:

$$COST(M) = \sum_{\text{all possible Ai}} T(A_i) \times \Pr(E(A_i) \mid \bigcup_i E(A_i)) \quad , \tag{1}$$

where $T(A_i)$ be the data transfer time required for A_i and $\Pr(E(A_i) \mid \bigcup_i E(A_i))$ is the probability that data will be transferred from the nodes indicated by A_i given that the data transfer request is successful. The goal of the FAP is to find the optimal allocation matrix M_{OP} such that $COST(M_{OP}) \le COST(M)$ for any valid allocation matrix M subject to the given allocation constraints (such as limited storage size).

The computational for this *FAP* is quite heavy. There are $n^{m \times k}$ possible allocation combinations for the m data files (each of which has k copies) on a n-node network. Since we assume that nodes are highly reliable, the probability that the program P will read data files from primary copies is very high. The cost function in (1) can be approached by

$$COST'(M) = T(A_1), \tag{2}$$

where A_1 indicates that all the data are transferred from primary copies.

Based on the simplified cost function *COST'*(*M*), algorithms used to find the optimal allocation matrix can finish within $O(n^m)$.

3. COMPUTATION OF *COST'*(*M*)

As stated in the introduction, the data transfer time (i.e. function *COST'*(*M*)) can be calculated using linear programming techniques, such as the simplex method. However, even if we reduce the scale of the problem to $O(n^m)$, solving $O(n^m)$ linear programming problems is still a tremendous task. This section presents an alternative way to compute *COST'*(*M*). Experimental results show that the proposed method is more efficient than the simplex method [1]. In addition, the information obtained using our method helps to further reduce the scale of the FAP.

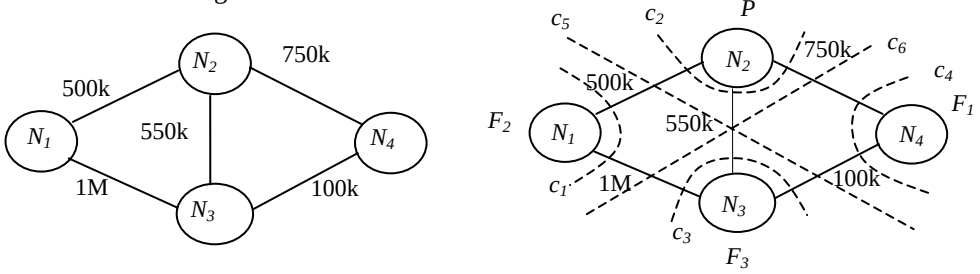
3.1 Critical Cuts

Let $(X, \overline{X})$ = the cut separating the nodes in set *X* (by letting the target node in *X*) from the other nodes,
let $cap(X, \overline{X})$ = capacity of the cut $(X, \overline{X})$, and
let L_V^A = the number of bytes from the data files allocated to *V* for a given allocation vector *A*, where *V* denotes a single node or a set of nodes.

Theorem 1. Let N_t be the target node, and let *S* be the set of source nodes. *T*(*A*), the time required for N_t to receive the data from *S* , can be computed by

$$\underset{i}{MAX}\left(\frac{L^A_{\overline{X_i}}}{cap(X_i, \overline{X_i})}\right)$$ for i = 1, 2, …,*h*, where $(X_i, \overline{X_i})$ (for i = 1, 2, …, *h*) are all the possible cuts that may separate N_t from *S* or a subset thereof (except the empty set).

Consider the DCS with the topology shown in Fig. 1(a). Assume that the program *P* is on N_2, and that the files F_1 ($|F_1|$ = 300k), F_2 ($|F_2|$ = 500k), and F_3 ($|F_3|$ = 1M) are on nodes N_4, N_1, and N_3, respectively. Fig. 1(b) shows all the possible cuts that may separate the source node(s) from the target node N_2. The corresponding parameters for Theorem 1 are listed in Table 1. We conclude that the shortest time for the program *P* to receive all the data is 1.30. The transmission speed is bounded by the capacity of cut c_5. We call the cut incurring the transmission bottleneck the *critical cut*.



(a) (b)

Fig. 1. (a) A simple DCS. (b)The cuts separate source nodes from the target node.

Table 1. Cuts and the corresponding parameters obtained by applying Theorem 1 to the DCS shown in Fig. 1.

cut	X	$\overline{X}$	$L_{\overline{X}}^{A}$	$cap(X, \overline{X})$	$L_{\overline{X}}^{A} / cap(X, \overline{X})$
c_1	$\{N_2,N_3,N_4\}$	$\{N_1\}$	$\|F_2\|$=500k	500k+1M=1500k	0.33
c_2	$\{N_2\}$	$\{N_1,N_3,N_4\}$	$\|F_1\|+\|F_2\|+\|F_3\|$=1800k	500k+550k+750k=1800k	1.00
c_3	$\{N_1,N_2,N_4\}$	$\{N_3\}$	$\|F_3\|$=1000k	1M+550k+100k=1650k	0.61
c_4	$\{N_1,N_2,N_3\}$	$\{N_4\}$	$\|F_1\|$=300k	750k+100k=850k	0.35
c_5	$\{N_2,N_4\}$	$\{N_1,N_3\}$	$\|F_2\|+\|F_3\|$=1500k	500k+550k+100k=1150k	1.30
c_6	$\{N_1,N_2\}$	$\{N_3,N_4\}$	$\|F_1\|+\|F_3\|$=1300k	1M+550k+750k=2300k	0.57

3.2 Reduction of Cuts

The enumeration of all the cuts as described in Theorem 1 is prohibitive for a large network. We have developed a theorem the purpose of for cut reduction. Prior to presenting this theorem, we need to introduce the concept of a cut-tree [8]. The cut-tree T is a flow-equivalent tree of the original network G. Each link in T represents a minimum cut of G. The n-node cut-tree shows the n-1 minimum cuts of G that do not cross each other. The following example illustrates the process to constructing a cut-tree.

Consider the original graph G shown in Fig. 1(a). Firstly, we may randomly choose two nodes, say N_1 and N_2, and perform the maximum flow computation. We get a minimum cut ($\{N_2, N_4\}$, $\{N_1, N_3\}$) with capacity 1150k. This is shown symbolically in Fig. 2(a). Next, we compute the maximum flow between N_1 and N_3. This computation may be done on a simpler graph (Fig. 2(b)) than G since we may consider nodes N_2 and N_4 as a single conceptual node. We get a minimum cut ($\{N_2, N_3, N_4\}$, $\{N_1\}$) with capacity 1500k. This is shown in Fig. 2(c). Note that in Fig. 2(c), node N_3 is attached to the conceptual node N_2, N_4 because they both are on the same side of the cut ($\{N_2, N_3, N_4\}$, $\{N_1\}$). Similarly, we calculate the maximum flow between N_2 and N_4 on the graph as shown in Fig. 2(d) and get a minimum cut ($\{N_1, N_2, N_3\}$, $\{N_4\}$) with capacity 850k. Fig. 2(e) shows the final cut-tree. Each link in the cut-tree corresponds to a cut in the original graph G. We show these cuts as dotted lines in Fig. 2(f). Since these cuts do not cross each other, we call them *partitioned cuts*.

Definition 1. Let $c_i = (X, \overline{X})$ and $c_j = (Y, \overline{Y})$ be two non-crossing cuts. If $X \subseteq Y$ (or equivalently, $\overline{X} \supseteq \overline{Y}$), then c_i is said to be an *ancestor* of c_j, and c_j is said to be a *descendant* of c_i. If $\overline{X} \cap \overline{Y} = \varnothing$, then c_i and c_j are said to be *brothers* of each other.

Definition 2. Cut c is called a *primary cut* if c is a partitioned cut, and if the capacity of c is smaller than the capacity of any ancestor cut of c.

Definition 3. Let $c_i = (X, \overline{X})$ and $c_j = (Y, \overline{Y})$ be two non-crossing cuts. The *exclusive-or* operation, denoted by $\oplus$, is defined as follows. If c_i and c_j have a brother relationship, then $c_i \oplus c_j \equiv (X \cap Y, \overline{X} \cup \overline{Y})$; if c_i and c_j have an ancestor/descendant relationship,

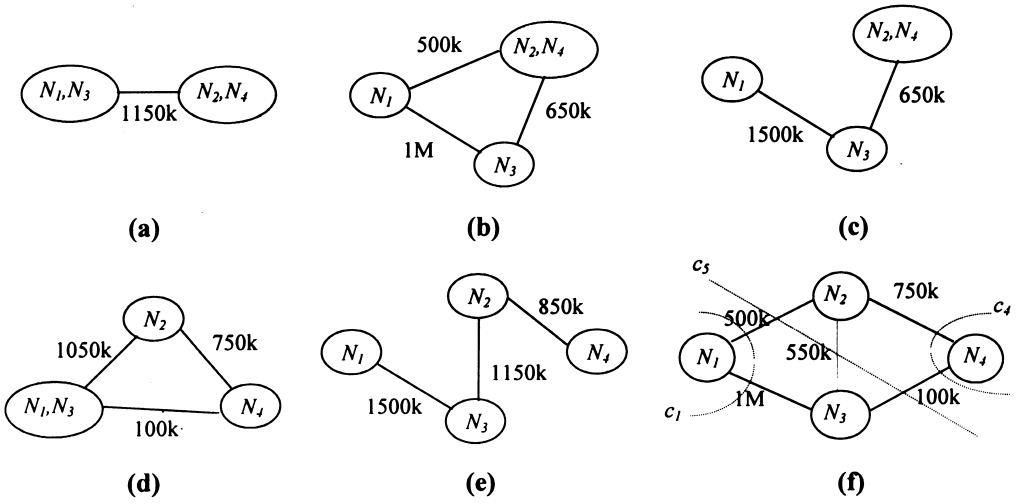


Fig. 2. An example to illustrate the process of obtaining a cut-tree. (a)-(d) the intermediate steps; (e) the final cut-tree; (f) the partitioned cuts.

and if c_i is an ancestor of c_j, then $c \oplus c_j \equiv (X \cup \overline{Y}, \overline{X} \cap Y)$.

Theorem 2. The critical cut can be factored into primary cuts based on exclusive-or operations, and none of these primary cuts is an ancestor (or descendant) of the others.

The result of Theorem 2 suggests that we may apply exclusive-or operations to primary cuts in order to generate a set of cuts, say *S*, which contains the critical cut. The members of *S* are called *possible critical cuts*. The procedure *Find_Possible_Critical_ Cuts* realizes the process of generating *S*. Since there are at most 2^p–1 possible critical cuts, the complexity of the procedure is $O(2^p)$, where *p* is the number of primary cuts.

PROCEDURE *Find-Possible-Critical-Cuts*

// *S* is the set of possible critical cuts. Initially, *S* is empty.
// *Q* is a queue of cuts. Initially, *Q* is empty.
Construct the cut-tree of the original network.
Decide on primary cuts.
Add all primary cuts to set *S*.
Enqueue all primary cuts to *Q*.
REPEAT
c = dequeue(*Q*)
FOR each of the primary cuts, c_P
Add and enqueue $c \oplus c_P$ to *S* and *Q*, respectively, unless
cond.1: *c* contains a factor cut which is an ancestor, or a descendent, of c_P.
cond.2: $c \cap c_P = \varnothing$ //i.e. cuts *c* and c_P have no common edges when they are presented by sets of edges.
END_FOR

UNTIL Q is empty
END_PROCEDURE

3.3 An Example

Consider the simple DCS shown in Fig. 1 as an illustrative example. The corresponding cut-tree and partitioned cuts are shown in Figs. 2 (e) and (f). Since the node N_2 is the target node, we let N_2 be the root of the cut-tree. Therefore, cuts c_4 and c_5 are primary cuts. Table 2 lists the steps needed to generate possible critical cuts. Three possible critical cuts are generated. The shortest transmission time can be calculated by

$$MAX(\frac{300}{850},\frac{1500}{1150},\frac{1800}{1800}) = \frac{1500}{1150} = 1.30$$

Table 2. Steps used to generate possible critical cuts (S: the set of possible critical cuts).

Cut to be checked	X	$\overline{X}$	Enter S?	Cut capacity	Remark
c_4	$\{N_1,N_2,N_3\}$	$\{N_4\}$	Yes	850k	Primary cut
c_5	$\{N_2,N_4\}$	$\{N_1,N_3\}$	Yes	1150k	Primary cut
$c_4 \oplus c_5 (=c_2)$	$\{N_2\}$	$\{N_1,N_3,N_4\}$	Yes	1800k	

4. FILE ALLOCATION ALGORITHMS

4.1 The NOFA (Near Optimum File Allocation Algorithm)

NOFA uses branch-and-bound to construct a search tree to obtain an allocation. The algorithm first determines the locations of primary copies by calling a procedure Sub_NOFA, which is responsible for constructing and traversing a search tree. Allocation of replicated copies for each data file is then carried out according to the locations of the primary copies. For each assignment of a replicated copy, a separate search tree is constructed and traversed. In these search trees, each node corresponds to the assignment of a particular file to a specific computer site. This assignment is presented in the form of an allocation vector. For example, the vector (3, 0, 1) means that F_1 is assigned to N_3, that F_2 has not been assigned, and that F_3 is assigned to N_1. Associated with each tree node (or allocation vector) is the evaluation function for the current assignment. In NOFA, there are two evaluation functions, $f(A)$ and $f^*(A)$, whose definitions are given below:

$f(A) = \underset{i}{MAX}\left(\frac{L^A_{X_i}}{Cap(X_i,\overline{X_i})}\right)$ for i = 1, 2, …, p, where $(X_i, \overline{X_i})$ (i=1 to p) are primary cuts;

$f^*(A) = \underset{i}{MAX}\left(\frac{L^{A}_{\overline{X_i}}}{Cap(X_i, \overline{X_i})}\right)$ for i = 1, 2, …, l, where $(X_i, \overline{X_i})$ (i=1 to l) are possible critical cuts.

Undoubtedly, f^* is a more precise evaluation function than f. However, the function f (with $O(n)$) is applied instead of f^* to evaluate an assignment for the sake of execution efficiency. The function f^* is computed only when a complete allocation is encountered (i.e., all the elements in the allocation vector are non-zero). It returns the accurate data transfer time.

To reduce the state spaces in the search tree, the following tree expansion rules are applied: (1) large files are assigned prior to small files; (2) among all the same-level nodes in the search tree, the one with the smallest f value is expanded first. Furthermore, the allocation of a file F_i to a node N_j is sometimes prohibited or unnecessary (see S1, S2, and S3 in the Procedure Sub_NOFA). Situations S1 and S2 are straightforward while S3 comes from the result of the following theorem.

Theorem 3. Let *A1* and *A2* be two allocation vectors which are only different in allocating a specific file, say F. A_1 allocates F to N_i while A_2 *allocates* F to N_j. If N_i is an ancestor node of N_j in the cut-tree, then $T(A_1) \leq T(A_2)$, where $T(A_1)$ and $T(A_2)$ denote the necessary data transfer time for A_1 and A_2, respectively.

ALGORITHM Optimal_File_Allocation

// K is the number of copies to be allocated for each file.

// $A_{op} \equiv (a_1, a_2, ..., a_m)$, $R \equiv (r_1, r_2, ..., r_m)$, $PFA \equiv (p_1, p_2, ..., p_m)$: allocation vectors.

// f^*_{op}, A_{op}, and $(Xop, \overline{Xop})$ are all global (or static) variables which record the current best allocation.

// M is an $m \times n$ (m: number of files; n: number of nodes) allocation matrix.

// *** Primary file allocation begin ***

$f^*_{OP} = \infty$, $FS = \{F_1, F_2, ..., F_m\}$, $R = (0, 0, ..., 0)$

CALL Sub_NOFA(R, FS)

$PFA = A_{OP}$

FOR i = 1 to m DO $M[i, p_i] = 1$

// *** Begin allocation of replicated copies ***

FOR k = 1 to K –1 DO

FOR i = 1 to m DO

$f^*_{OP} = \infty$, FS = $\{F_i\}$, R = ($r_1 = p_1$, $r_2 = p_2$, ..., $r_i = 0$, ..., $r_m = p_m$)

CALL Sub_NOFA(R,FS)

$M[i, a_i] = 1$

END_FOR

END_FOR

END_ALGORITHM

PROCEDURE Sub_NOFA(*v*,*FS*)
IF *FS* is empty THEN
Find the critical cut $(X, \overline{X})$ and calculate $f^*(A_v)$, where A_v is the associative allocation of v
IF $f^*(A_v) < f^*_{OP}$ THEN $A_{OP} = A_v$, RETURN
END_IF
Remove F_{max} , the largest-sized file, from the set *FS*
Expand the node v by assigning F_{max} to each of the processing sites, say N_j , except that
S1: Allocation of F_{max} to N_j is disallowed by the allocation constraints,
S2: N_j has a copy (or copies) of F_{max},
S3: N_j has an ancestor node (in the cut-tree) which are not in the previous cases.
Determine the expansion order of all successor nodes according to their f values.
FOR each of the successor nodes, say u
IF $f(A_u) < f^*_{OP}$ AND $L\frac{A_u}{X_{OP}} < L\frac{A_{OP}}{X_{OP}}$ THEN CALL Sub_NOFA(u,*FS*)
END_FOR
END_PROCEDURE

4.2 An Example

Consider the network shown in Fig. 1(a). Node N_2 is assumed to be the target node, and three data files, F_1 (300k bytes), F_2 (500k bytes), and F_3 (1M bytes), are to be allocated. Each data file has two copies, but none of them can be allocated to the target node. Except for this, there is no other allocation constraint. The primary file allocation results in the following trace (see Fig. 3).

- Initially, the root $v_0 = (0, 0, 0)$.
- Expand v_0 by allocating file F_3 (the largest data file) to get v_1 (with $f = 0.87$) and v_2 (with $f = 1.18$). Node v_1 is selected for the next expansion since it has a smaller f value. Repeat this process, and nodes v_3, v_4, v_5 , and v_6 will be built sequentially.
- Expand v_6 (with $f = 0.94$) and obtain v_7. Compute f^*(= 1.00) and obtain the critical-cut $c_2 = (\{N_2\}, \{N_1, N_3, N_4\})$. As a result, $f^*_{OP} = 1.00$; $A_{OP} = (4, 4, 3)$; $(Xop, \overline{Xop}) = (\{N_2\}, \{N_1, N_3, N_4\})$. Return to the higher level.
- Since the f value of v_5 is not less than f^*_{OP}, return to the higher level. Similarly, the f values of v_3 and v_2 are not less than f^*_{OP}, so return to the caller, NOFA.
- Update M as $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$, and the allocation of primary copies completes.

Assignment of the second copy of *F1* has the state space shown in Fig. 4(a).

- $v_0 = (0, 4, 3)$.
- Expand v_0 by allocating file F_1 to get v_1 .

- Expand v_1 (with f = 1.13) and obtain v_2. Compute f^*(= 1.13) and obtain the critical-cut $c_5 = (\{N_2, N_4\}, \{N_1, N_3\})$ and $A_{OP} = (3, 4, 3)$. Return to the higher level.
- Return to NOFA and update M as $\begin{bmatrix} 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

Assignment of the second copies of *F2* and *F3* is carried out in a similar way (see Fig. 4(b) and Fig. 4(c)). Finally, we obtain an allocation matrix: $\begin{bmatrix} 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 1 \\ 2 & 0 & 1 & 0 \end{bmatrix}$.

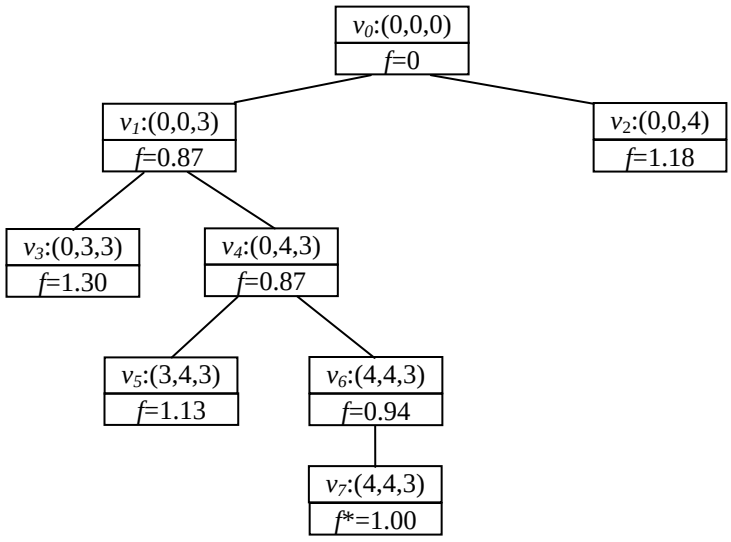


Fig. 3. State space tree for allocation of primary copies.

Table 3 lists the probabilities and data transfer times for all the possible node status combinations by assuming that all the nodes have the same probability (or reliability) of success, *Pr*. The probabilities of successful execution of program *P* are 0.981 and 0.999801, and the expected data transfer times (given that execution of *P* is successful) are 1.052 and 1.006, for *Pr* = 0.9 and *Pr* = 0.99, respectively.

4.3 The HFA (Heuristic File Allocation Algorithm)

A polynomial time complexity algorithm is presented in this subsection. HFA is similar to NOFA except that HFA calls the procedure Sub_HFA instead of expanding the search tree. In Sub_HFA, the searching process stops as soon as the first complete allocation is encountered.

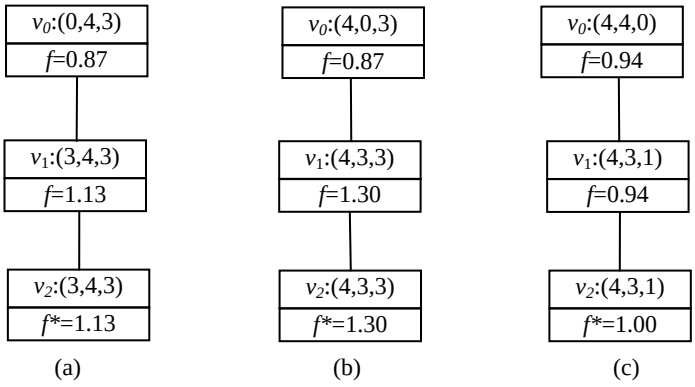


Fig. 4. State space tree for allocation of second copies.

Table 3. List of probabilities and data transfer times for different node status combinations. (0: failure, 1: success, *Pr*: node reliability).

N_1	N_3	N_4	Execution	Probability	Pr = 0.9	Pr = 0.99	Data Transfer Time
0	0	0	0	$(1-Pr)^3$	0.001	0.000001	-
0	0	1	0	$Pr(1-Pr)^2$	0.009	0.000099	-
0	1	0	1	$Pr(1-Pr)^2$	0.009	0.000099	1.57
0	1	1	1	$Pr^2(1-Pr)$	0.081	0.009801	1.00
1	0	0	0	$Pr(1-Pr)^2$	0.009	0.000099	-
1	0	1	1	$Pr^2(1-Pr)$	0.081	0.009801	1.00
1	1	0	1	$Pr^2(1-Pr)$	0.081	0.009801	1.57
1	1	1	1	Pr^3	0.729	0.970299	1.00

PROCEDURE Sub_HFA(*v*,*FS*)
current_node = *v*
REPEAT
Remove F_{max} , the largest-sized file, from the set *FS*
Expand current_node by assigning F_{max} to each of the processing sites, say N_j , except that
S1: allocation of F_{max} to N_j is disallowed by the allocation constraints,
S2: N_j has a copy (or copies) of F_{max},
S3: N_j has an ancestor node (in the cut-tree) which does not exist in the previous cases.
Let current_node be the node having the smallest *f* value among all the successor nodes.
UNTIL *FS* is empty

Return current_node
END_PROCEDURE

5. PERFORMANCE

Herein, the performance of NOFA and HFA has been evaluated by means of a simulation program which allocates five files of randomly generated sizes on a network of ARPA net topology (21 nodes, 26 links). The target node and the available link capacities are randomly generated. The allocations generated by NOFA and HFA are evaluated by comparing their corresponding expected data transfer times with that of the optimal allocation. However, since obtaining the optimal allocation is not computationally feasible (especially in the case of allocation of multiple replicated copies), we obtain the optimal allocation by selecting the best one among 100,000 random allocations and the allocations found by NOFA and HFA. Experiments have been conducted with K = 1 and 2, respectively, where K is the number of copies of each data file. The results are shown in Fig. 5, where the *X*-axis is the node reliability and the *Y*-axis is the ratio $COST(M_{OP})/ COST(M)$, where M is the allocation found by NOFA or HFA and M_{OP} represents the optimal allocation.

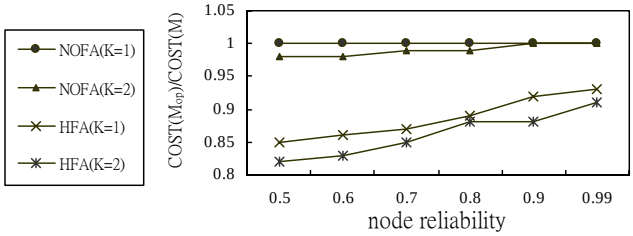


Fig. 5. Experimental results obtained using NOFA and HFA.

6. CONCLUSIONS

We have developed a flow-based data transmission model and presented a novel method for solving it. We have also presented two algorithms, NOFA and HFA, for dealing with the file allocation problem. Simulation results show that a very high percentage of the allocations generated by NOFA are optimal allocations, especially when the nodal reliability is high. The results also suggest that the heuristic approach, HFA, is a good approximation.

ACKNOWLEDGEMENTS

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract no. NSC872213E009094.

REFERENCES

1. P. Y. Chang and D. J. Chen, "Optimal routing for distributed systems with data replication," in *Proceeding of IEEE International Computer Performance & Dependability Symposium*, 1996, pp. 42-51.
2. P. S. Chen and J. Akoka, "Optimal design of distributed information systems," *IEEE Transactions on Computers*, Vol. C-29, No.12, 1980, pp. 1068-1080.
3. R. S. Chen, D. J. Chen, and Y. S. Yeh, "Reliability optimization of distributed computing systems subject to capability constraints," *An International Journal of Computer & Mathematic with Application*, Vol. 29, No. 4, 1995, pp. 93-99.
4. W. Chu, "Optimal file allocation in a computer network," *Computer-Communication Systems*, Prentice-Hall, Englewood Cliffs, N. J, 1973, pp. 82-94.
5. L. W. Dowdy and D. V. Foster, "Comparative models of file assignment problem," *ACM Computer. Surveys*, Vol. 14, No. 2, 1982, pp. 287-313.
6. A. E. El-Abd, "Modeling resources allocation and performance measures in distributed computer networks," in *Proceeding of IEEE Singapore International on Networks/ Conference on Information Engineering*, 1995, pp. 581-586.
7. V. Foster, L. W. Dowdy, and J. E. Ames, "File assignment in a computer network," *Computer Network*, Vol. 5, No. __, 1981, pp. 341-349.
8. T. C. Hu, *Combinatorial Algorithms*, Addison-Wesley, Menlo Park (Calif.), 1986, pp. 60-83.
9. J. F. Kurose and R. Simha, "A microeconomic approach to optimal resource allocation in distributed computer systems," *IEEE Transactions on Computers*, Vol. C-38, No. 5, 1989, pp. 705-717.
10. S. Mahmoud and J. S. Riordon, , "Optimal allocation of resources in distributed information networks," *ACM Transactions on Database System*, Vol. 1, No. 1, 1976, pp. 66-78.
11. L. Morgan and K. D. Levin, "Optimal program and data locations in computer networks," *Communication of ACM* , Vol. 20, No. 5, 1977, pp. 315-322.
12. R. M. Pathak, A. Kumar, and Y. P. Gupta, , "Reliability oriented allocation of files on distributed systems," in *Proceedings of IEEE Symposium on Parallel and Distributed Processing*, pp. 886-893.
13. S. Ram and R. E. Marsten, "A model for database allocation incorporating a concurrency control mechanism," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 3, No.__, 1991, pp. 389-395.
14. V. Ramamoorthy and K. M. Chandy, "Optimization of memory hierarchies in multiprogrammed systems," *Journal of ACM* , Vol. 17, No. 3, 1970, pp. 426-445.

Pao-Yuan Chang (**張寶源**) is an associate professor of Information Management at the Ta Hwa Institute of Technology (Hsinchu, Taiwan). He holds a B.S. degree in Computer Engineering from National Chiao Tung University (Hsinchu, Taiwan), an M.S. degree in Computer Science from University of Missouri (USA), and a Ph.D degree in Computer Science and Information Engineering from National Chiao Tung University. His research interests include performance and reliability analysis of distributed systems.

Deng-Jyi Chen (陳登吉) received the B.S. degree in Computer Science from Missouri State University (cape Girardeau), and the M.S. and Ph.D degrees in Computer Science from the University of Texas (Arlington) in 1983, 1985, and 1988, respectively. He is now a professor at National Chiao Tung University (Hsinchu, Taiwan). Prior to joining the faculty of National Chiao Tung University, he was with National Cheng Kung University (Tainan, Taiwan). He has published more than 100 referred journal and conference papers in the areas of performance and reliability modeling and evaluation of distributed systems, computer networks, fault-tolerant systems, software reuse, and object-oriented systems. He has also been the chief leader of several commercial products, some of with are now marketed around the world. He has also received the research award yearly from the National Science Council Taiwan for the last seven years and served as a committee member for several academic and industrial organizations.

Krishna M. Kavi is currently a professor and eminent scholar of Computer Engineering. Prior to joining UAH, he was a professor of Computer Science and Engineering at the University of Texas at Arlington. For two years (1993-1995) he was a program manager at the National Science Foundation, managing operating systems, programming languages and compiler programs in CCR the Division. He was an IEEE Computer Society (CS) Distinguished Visitor (1989-91), editor of IEEE Transactions on Computers (1993-1997), and editor of the Computer Society Press (1987-1991). His primary research interest lies in computer systems architecture, including dataflow and multithreaded systems, operating systems, and compiler optimization. His other research interests include formal specifications of concurrent processing systems, performance modeling and evaluation, load balancing and scheduling of parallel programs. He has published over 100 technical papers on these topics. He received his B.E. (Electrical) degree from the Indian Institute of Science, and MS and Ph.D. (Computer Science and Engineering) degrees from Southern Methodist University. He is a senior member of IEEE and a member of ACM.