Multimedia File Allocation on VC Networks Using Multipath Routing

Pao-Yuan Chang,
Deng-Jyi Chen, *Member*,
IEEE Computer Society, and
Krishna M. Kavi, *Senior Member*, *IEEE*

Abstract—The problem of allocating high-volume multimedia files on a virtual circuit network with the objective of maximizing channel throughput (and minimizing data transmission time) is addressed. The problem is formulated as a multicommodity flow problem. We present both the optimal and suboptimal solutions to the problem using novel approaches.

Index Terms—File allocation, virtual circuit, cut, maximum flow, linear programming, branch and bound.

1 Introduction

As web-based computing is becoming commonplace, timely access to data files that are distributed over a large number of nodes becomes critical. This in turn requires optimal allocation of data files to nodes to improve the data transfer time. The class of problems dealing with the assignment of files to processing nodes to optimize performance is commonly known as the file allocation problem (FAP). This class of problems with different objectives and constraints has been addressed in the literature [1], [2], [3], [5], [7], [10], [12], [14], [15], [17], [18], [19]. Since FAPs are complex integer programming problems with no known efficient solutions [4], heuristics are used to find optimal solutions.

In this paper, we address FAP in a virtual circuit network environment. A virtual circuit enables bandwidth to be reserved throughout the connection [6], [9], [11]. We assume a multipath routing that allows multiple routes for simultaneous data transmission such that a high probability of successful connection for on-demand high-bandwidth multimedia data transfer can be achieved. We also assume that the combined size of the files is too large to fit on any single node. Under these assumptions, the FAP can be described as follows: Given the location of computing nodes, data files needed for the computation, their sizes, available link capacities, and other allocation constraints, allocate the files to nodes to achieve minimum time for data transfer. In a high bandwidth network with well-behaved traffic, the data transfer time for a file is primarily determined by transmission throughput rather than by link delays (including queuing delay, propagation delay, and transmission time). The objective for our problem is the allocation of files to achieve maximum overall data transfer rate.

However, unlike other FAPs, such an objective cannot be represented by a simple linear or nonlinear function. The objective function itself is a multicommodity flow problem and finding a feasible solution requires the use of linear programming (LP) techniques. A simplistic approach to finding an optimal solution formulates and solves every corresponding LP subproblem for every possible valid file allocations. This approach is prohibitively expensive in terms of computational cost. We present an alternate method that uses the critical cut method (instead of the LP) to compute the objective function. An algorithm, OFA (Optimal File Allocation Algorithm), based on branch-and-bound with the evaluation functions derived from critical cut concept is developed in finding the optimal allocation. While this approach is more efficient than the exhaustive search of LP, the optimal solution still requires exponential time. We then present a polynomial-time heuristic algorithm HFA (Heuristic File Allocation Algorithm).

- *P.-Y. Chang is with the Department of Information Management, Ta Hwa Institute of Technology, Hsin Chu, 30700, Taiwan. E-mail: pychang@et4.thit.edu.tw.*
- *D.-J. Chen is with the Computer Science and Information Engineering Department, National Chiao Tung University, Hsin Chu, 30050 Taiwan. E-mail: djchen@csie.nctu.edu.tw.*
- *K.M. Kavi is with the Department of Electrical and Computer Engineering, University of Alabama at Huntsville, 301 Sparkman Dr., Room EB267, Huntsville, AL 35899. E-mail: Kavi@ece.uah.edu.*

Manuscript received 13 July 1998; revised 19 May 1999; accepted 14 Apr. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 107142.

2 Problem Formulation

We assume that a distributed computing system can be described as an undirected graph with nodes representing computing sites and edges representing communication links. We assume virtual circuits and multipath routing for communication between a source and a destination. Intermediate nodes do not buffer packets, but simply send all received packets immediately. Link propagation delay is assumed negligible. We assume that the combined size of the files is too large to fit on any single node and a file cannot be split for distribution among multiple sites. A communication path is necessarily acyclic.

In a packet-switched subnet, files are broken into packets for transmission. In general, the size of a packet is much smaller than that of a multimedia file. A flow model can therefore accurately reflect the data transmission behavior. Let M be the number of data files to be allocated and N be the number of computer sites in the network. We label the data files with $F_1, F_2, \ldots, F_M$ and the computer sites with $1, 2, \ldots, N$, respectively. An allocation $A = [A_{ij}]$ of the data files is defined as an $M \times N$ matrix, where

$$A_{ij} = \begin{cases} 1 & \text{if data file } F_i \text{ is allocated to node } j \\ 0 & \text{otherwise.} \end{cases}$$

The nodes on which data files are allocated are referred to as the *source nodes*. The computing node (which accesses files) is referred to as the *target node*. In our case, the objective function is the shortest time required to transfer data from the source nodes to the target node. For a given allocation A, the objective function $COST(A)$ can be described by the following formulation:

$$\begin{aligned} &MinimizeMax\left\{\frac{|x_{ij}|}{cap_{ij}} \middle| i > j, j = 1, 2, 3, \ldots, N-1, cap_{ij} > 0\right\} \\ &subject to \sum_{i=j}^{N} x_{ij} - \sum_{k=1}^{j} x_{jk} \\ &= \begin{cases} -L_j(A) & \text{if node } j \text{ is a source node} \\ L & \text{if node } j \text{ is the target node} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } j = 1, 2, \ldots, N \\ &x_{ij} = 0 \text{ if nodes } i \text{ and } j \text{ are not adjacent,} \\ &\qquad \text{for all } i > j, j = 1, 2, \ldots, N-1, \end{aligned} \tag{1}$$

where

$cap_{ij} =$ link capacity between nodes i and j,

$x_{ij} =$ a variable indicating the number of bytes to be transferred from node i to its adjacent node j (note, if the solution to the variable x_{ij} is negative, then data flows from nodes j to i),

$L_j(A) =$ the number of bytes assigned to node j for allocation A,

$L =$ the number of bytes in all data files.

The constraints in (1) stem from a balanced inflow and outflow should. The term $|x_{ij}|/cap_{ij}$ in (1) is the time for transmitting data

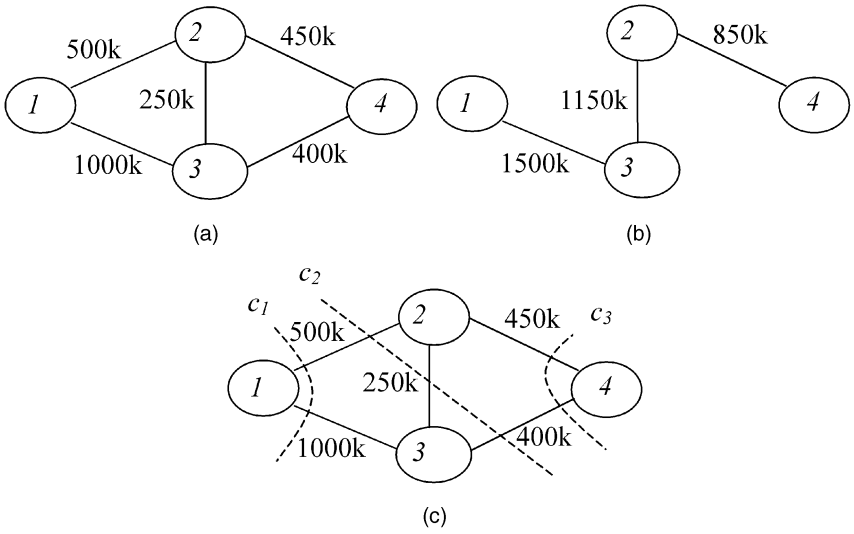

Fig. 1. (a) The original network G. (b) The cut-tree of G. (c) The partitioned cuts.

from nodes i to j, or from nodes j to i, depending on the sign of x_{ij}. The link having the maximum value of $|x_{ij}|/cap_{ij}$ is the bottleneck. Since the inherent link delay was ignored, the actual transfer time is limited by the time required to transfer data over the bottleneck link. The objective is to minimize the data transfer time through this link.

Our FAP is thus expressed as

$$Minimize\ COST(A)\ subject\ to \sum_{j=1}^{N} A_{ij} = 1 \quad \text{for } i = 1, 2, \cdots, M$$

and any other constrains (e.g., limited storage size).

3 Computation of $COST(A)$

3.1 Critical Cut

COST(A) as formulated in (1) can be transformed into an LP problem and then solved. In this section, we present an alternative, the critical cut method that is based on cut-sets of an undirected graph, to compute COST(A). Proofs of theorems and lemmas associated with our method are not included in this paper due to space limitations.

Notation

e_{ij} Edge connecting nodes i and j.

$(X, \overline{X})$ The cut separating the nodes in set X (with target node in X) from the other nodes.

$cap(X, \overline{X})$ Capacity of the cut $(X, \overline{X})$.

$L_V(A)$ The number of bytes from the data files allocated to V in an allocation A, where V denotes a single node or a set of nodes.

Definition 1. *Let S be the set of source nodes for an allocation A and $C = \{(X_1, \overline{X_1}), (X_2, \overline{X_2}), \ldots, (X_m, \overline{X_m})\}$ be the set of cuts that separate the target node from S or a subset thereof (except the empty set). Cut $(X_t, \overline{X_t})(\in C)$ is called a* critical cut *if*

$$\frac{L_{\overline{X_t}}(A)}{cap(X_t, \overline{X_t})} \geq \frac{L_{\overline{X_i}}(A)}{cap(X_i, \overline{X_i})}$$

for $i = 1, 2, \ldots, m$.

Theorem 1. *The shortest time required for the target node to receive the data from source nodes can be computed by $\frac{L_{\overline{X_t}}(A)}{cap(X_t, \overline{X_t})}$, where $(X_t, \overline{X_t})$ is a critical cut.*

The application of Theorem 1 itself is inefficient since it involves the enumeration of all cuts in set C. We now present an approach that reduces the number of cuts that must be examined in finding the critical cut. The cut reduction is based on the concept of cut-tree proposed by Hu [8]. Consider the example in Fig. 1. Fig. 1a shows the original network, G, along with link capacities. Fig. 1b shows the cut-tree generated by Hu's algorithm. Fig. 1c illustrates that each link of the cut-tree represents a minimum cut of G. Since these minimum cuts do not cross each other, we call them the *partitioned cuts*.

Definition 2. *Let $c_i = (X, \overline{X})$ and $c_j = (Y, \overline{Y})$ be two cuts that do not cross. If $X \subseteq Y$ (or, equivalently, $\overline{X} \supseteq \overline{Y}$), c_i is said to be an* ancestor *of c_j and c_j a descendant of c_i. If $\overline{X} \cap \overline{Y} = \emptyset$, c_i and c_j are said to be* siblings *of each other.*

The relationship between any pair of partitioned cuts can be easily identified by the corresponding cut-tree. Let l_i and l_j be the links in the cut-tree corresponding to the partitioned cuts c_i and c_j. By letting the target node be the root of the cut-tree, if there is a path from the root to a leaf containing l_i and l_j, then c_i and c_j have an ancestor/descendent relationship. Cut c_i is the ancestor of c_j if l_i is closer to the root than l_j is in this path. If no such a path can be found, then c_i and c_j have a sibling relationship.

Definition 3. *Let R be the set of partitioned cuts. Cut $c \in R$ is called a* primary cut *if, except c itself, there is no ancestor cut of c can be found in R, or there are ancestor cuts of c in R, but the capacity of c is less than that of any of its ancestor.*

Definition 4. *Edge e in the network is called a* primary edge *if at least one primary cut contains edge e; otherwise, edge e is called a* nonprimary edge.

For example, in Fig. 1, if node 2 is the target node, then $c_2 \equiv (\{2,4\}, \{1,3\})$ is an ancestor of $c_1 \equiv (\{2,3,4\}, \{1\})$; $c_3 \equiv (\{1,2,3\}, \{4\})$ is a sibling of c_1 and c_2. Cuts c_2 and c_3 are primary cuts, but cut c_1 is not. All edges except e_{13} are primary edges.

Lemma 1. *A critical cut contains only primary edges.*

However, an arbitrarily collection of primary edges is not necessarily a cut. To effectively find the cuts containing only primary edges, we have the following definition.

Definition 5. *Let $c_i = (X, \overline{X})$ and $c_j = (Y, \overline{Y})$ be two noncrossed cuts. The* exclusive-or *operation, denoted by $\oplus$, is defined as follows: If c_i*

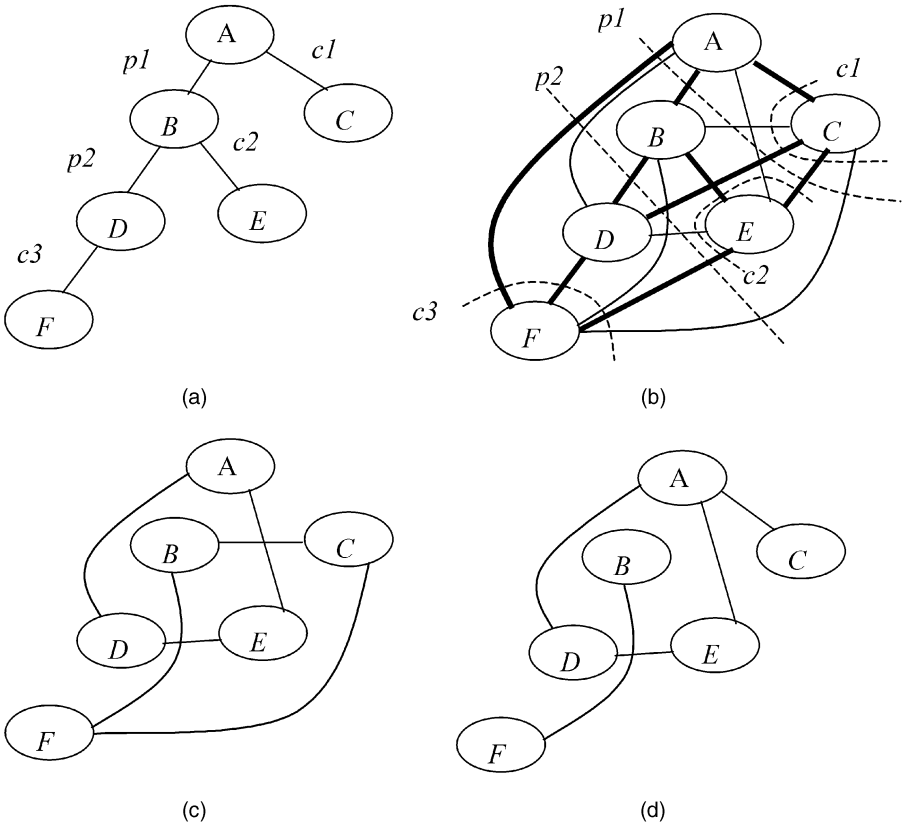

Fig. 2. (a) The cut-tree presenting the relationships among $p1$, $p2$, $c1$, $c2$, and $c3$. (b) The reduced graph, where the edges presented by bold lines is the cut $p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3$. (c) The graph obtained by removing the cut $p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3$ (note that a cut is a set of edges). (d) The graph obtained by removing the cut $\{E_{AB}, E_{AF}, E_{BC}, E_{BD}, E_{BE}, E_{CD}, E_{CE}, E_{CF}, E_{DF}, E_{EF}\}$.

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

The following properties of exclusive-or operation can be easily proven.

- If cuts are represented by sets of edges, then, for two noncrossed cuts c_i and c_j, $c_i \oplus c_j = c_i \cup c_j - c_i \cap c_j$.
- Exclusive-or operation is commutative and associative.
- $c_i \oplus c_j$ is also a cut.
- If c_i and c_j are siblings, $c_i \oplus c_j$ is an ancestor of both c_i and c_j.
- If c_i is an ancestor of both c_j and c_k, then c_i is also an ancestor of $c_j \oplus c_k$.
- If c_i is a sibling of both c_j and c_k, then c_i is also a sibling of $c_j \oplus c_k$.

Lemma 2. *Any cut in a network can be factored into partitioned cuts based on exclusive-or operations.*

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

Proof. We first prove that a critical cut can be factored into primary cuts (Part 1) and then prove that any pair of these primary cuts are not ancestor/descendant of each other (Part 2).

The proof to Part 1 is derived from Lemmas 1 and 2. Lemma 1 states that a critical cut contains only primary edges. Therefore, we can combine adjacent nodes connected by nonprimary edges into a super node. We denote the combined network G^*. Since there is no nonprimary edge in G^*, all the partitioned cuts in G^* are primary. Thus, Part 1 follows immediately from Lemma 2.

The proof to Part 2 is as follows: Let cut c be the result of applying exclusive-or operations on a set of primary cuts, S, which contains a pair of cuts having ancestor/descendant relationship. If there is more than a pair of primary cuts in S having ancestor/descendant relationship, we may select a cut pair $p1 = (V, \overline{V})$ and $p2 = (W, \overline{W})$ for consideration, where $p1$ is the only ancestor cut of $p2$ in S. Such a selection is always feasible since, for example, assuming that $c = \hat{p}_1 \oplus \hat{p}_2 \oplus \hat{p}_3 \oplus \hat{p}_4 \oplus \hat{p}_5$ and $\hat{p}_1$ is an ancestor of $\hat{p}_2$, $\hat{p}_2$ is an ancestor of $\hat{p}_3$, and $\hat{p}_4$ and $\hat{p}_5$ are siblings of $\hat{p}_1$, we thus choose the pair $\hat{p}_1$ and $\hat{p}_2$, rather than other ancestor/descendent pairs: $\hat{p}_1$ and $\hat{p}_3$, $\hat{p}_2$ and $\hat{p}_3$, for consideration. The other members of S, according to their relationships to $p1$ and $p2$, can be grouped into three distinct sets: $S1$, $S2$, and $S3$, where $S1$ is the collection of $p1$'s sibling cuts, $S2$ is the collection of cuts which are not only $p1$'s descendants but also $p2$'s siblings, and $S3$ is the set of $p2$'s descendants. Let $c1 = (X, \overline{X})$, $c2 = (Y, \overline{Y})$, and $c3 = (Z, \overline{Z})$ be the results of applying exclusive-or operations on $S1$, $S2$, and $S3$, respectively. Cut c thus equals to $p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3$. According to the properties of exclusive-or operation, cut $c1$ is $p1$'s sibling, cut $c2$ is $p2$'s sibling and $p1$'s descendant, and cut $c3$ is $p2$'s descendant. The relationships among cuts $p1$, $p2$, $c1$, $c2$, and $c3$ can be clearly observed in Fig. 2a. It is obvious that Fig. 2a is a cut-tree of Fig. 2b, which is reduced from the original graph by letting the nodes $A = V \cap X$, $B = W \cap \overline{V} \cap Y$, $C = \overline{X}$,

Cuts	X	$\overline{X}$	Enter S?	Capacity	Remark
c_1	{2,3,4}	{1}	Yes	1500k	Primary cut
c_2	{1,3}	{2,4}	Yes	1150k	Primary cut
c_3	{1,2,3}	{4}	Yes	850k	Primary cut
$c_1 \oplus c_2$	{3}	{1,2,4}	Yes	1650k	
$c_1 \oplus c_3$	{2,3}	{1,4}	No		$c_1 \cap c_3 = \emptyset$
$c_2 \oplus c_3$	{1,3,4}	{2}	No		c_3 is descendant of c_2
$c_3 \oplus (c_1 \oplus c_2)$	{3,4}	{1,2}	No		c_3 is descendant of c_2

Fig. 3. Steps in generating possible critical cuts (S: set of possible critical cuts).

$D = Z \cap \overline{W}$, $E = \overline{Y}$, and $F = \overline{Z}$, and the capacities of edges E_{IJ} for $I, J \in \{A, B, C, D, E, F\}$ be given by $\sum_{node i \in I,\ node\ j \in J} cap(e_{ij})$. If a cut is presented by a set of edges,

$$p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3 = \{E_{AB}, E_{AC}, E_{AF}, E_{BD}, E_{BE}, E_{CD}, E_{CE}, E_{DF}, E_{EF}\}$$

is a cut separate $\{A, D, E\}$ from $\{B, C, F\}$ (see Fig. 2c). The bold lines in Fig. 2b show these edges. Consider cut

$$p1 = \{E_{AB}, E_{AD}, E_{AE}, E_{AF}, E_{BC}, E_{CD}, E_{CE}, E_{CF}\}$$

and the cut

$$\{E_{AB}, E_{AF}, E_{BC}, E_{BD}, E_{BE}, E_{CD}, E_{CE}, E_{CF}, E_{DF}, E_{EF}\}$$

(see Fig. 2d). Both cuts separate A from B, but cut $p1$ is known as the minimum cut. Thus, we have

$$\begin{aligned} &cap(E_{AB}) + cap(E_{AD}) + cap(E_{AE}) + cap(E_{AF}) + cap(E_{BC}) \\ &+ cap(E_{CD}) + cap(E_{CE}) + cap(E_{CF})) \leq \\ &cap(E_{AB}) + cap(E_{AF}) + cap(E_{BC}) + cap(E_{BD}) + cap(E_{BE}) \\ &+ cap(E_{CD}) + cap(E_{CE}) + cap(E_{CF}) + cap(E_{DF}) + cap(E_{EF}) \end{aligned} \quad (2)$$

or, equivalently,

$$\begin{aligned} &cap(E_{AD}) + cap(E_{AE})) \leq \\ &cap(E_{BD}) + cap(E_{BE}) + cap(E_{DF}) + cap(E_{EF}). \end{aligned} \quad (3)$$

On adding $cap(E_{AB}) + cap(E_{AC}) + cap(E_{AF})$ to the lefthand side of (3) and $cap(E_{AB}) + cap(E_{AC}) + cap(E_{AF}) + cap(E_{CD}) + cap(E_{CE})$ to the righthand side of (3), we obtain

$$\begin{aligned} &cap(E_{AB}) + cap(E_{AC}) + cap(E_{AD}) + cap(E_{AE}) + cap(E_{AF}) \leq \\ &cap(E_{AB}) + cap(E_{AC}) + cap(E_{AF}) + cap(E_{BD}) + cap(E_{BE}) \\ &+ cap(E_{CD}) + cap(E_{CE}) + cap(E_{DF}) + cap(E_{EF}). \end{aligned} \quad (4)$$

The lefthand side of (4) is the capacity of the cut $p1 \oplus c1$ (i.e., the cut $(\{A\}, \{B, C, D, E, F\})$) and the righthand side of (4) is the capacity of cut $p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3$ (i.e., the cut $(\{A, D, E\}, \{B, C, F\})$). Thus, the cut $p1 \oplus c1$ is more critical than the cut obtained by $p1 \oplus p2 \oplus c1 \oplus c2 \oplus c3$ (i.e., cut c). Cut c is thus not critical. This proves Part 2 and Theorem 2 thus follows. □

3.2 Algorithm

Theorem 2 suggests that we may apply exclusive-or operations on primary cuts in order to generate a set of cuts, say S, which contains the critical cut. The members in S are referred to as the *possible critical cuts*. The following procedure, Find_Possible_Critical_Cuts realizes the process of generating possible critical cuts.

```
PROCEDURE Find-Possible-Critical-Cuts
/* S is the set of possible critical cuts. */
/* Initially S is empty. Finally, S holds the possible critical cuts. */
/* Q is a queue of cuts. Initially Q is empty. */
   Construct the cut-tree of the original network.
   Decide 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 ⊕ 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 ∩ c_P = ∅
      END_FOR
   UNTIL Q is empty
END_PROCEDURE
```

3.3 Example

Consider the network in Fig. 1a. Files sizes and distribution are given as follows: Node 3 is the computing node (i.e., target node). File F_1 ($|F_1| = 300$k) is on node 2; file F_2 ($|F_2| = 400$k) is on node 1; file F_3 ($|F_3| = 800$k) is on node 4. By letting node 3 be the root of the cut-tree, we observe that all partitioned cuts are primary cuts. Fig. 3 lists the steps in generating possible critical cuts. Four possible critical cuts are generated. The shortest data transfer time can be calculated by

$$MAX\left(\frac{400}{1500}, \frac{1100}{1150}, \frac{800}{850}, \frac{1500}{1650}\right) = \frac{1100}{1150} = 0.96.$$

4 Optimal File Allocation Algorithms

4.1 Algorithm

In this section, we present an algorithm, OFA (Optimal File Allocation algorithm), which uses branch & bound techniques for constructing a search tree and obtaining the optimal allocation for out FAP. Each tree node corresponds to the assignment of a particular file to a specific computer site. The internal nodes of the search tree correspond to partial allocations; the leaf nodes represent complete allocations. Associated with each node in the search tree is the evaluation function for the current assignment. To ensure that the searching always yields the optimal solution, the evaluation function for the internal nodes should underestimate the path cost [20]. In the OFA, the evaluation function for a current allocation A, denoted by $f(A)$, is defined as

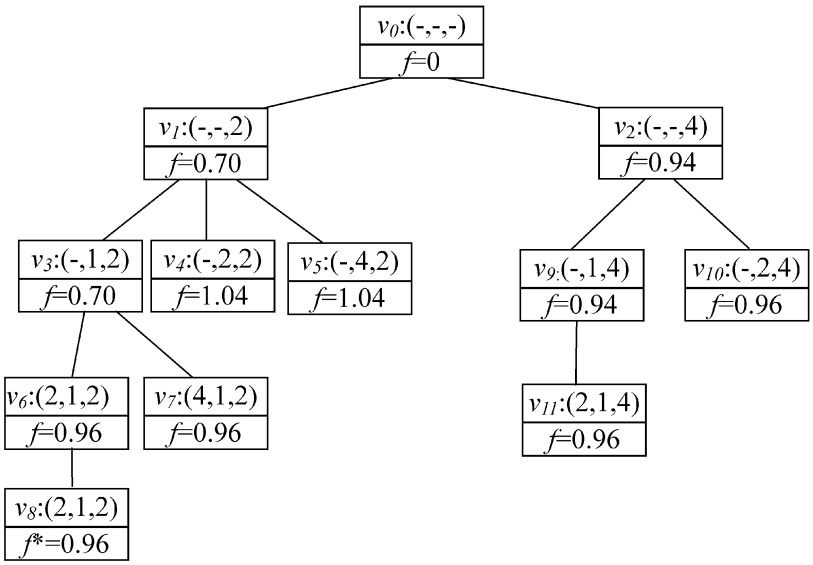

Fig. 4. The state space tree generated by OFA applied to the example in Section 4.

$$f(A) = \underset{i}{Max}\left(\frac{L_{\overline{X_i}}(A)}{Cap(X_i, \overline{X_i})}\right) \qquad \text{for } i = 1, 2, \ldots, k,$$

where $(X_1, \overline{X_1}), (X_2, \overline{X_2}), \ldots, (X_k, \overline{X_k})$ are primary cuts.

The actual cost of the current allocation A, denoted by $f^*(A)$, can be computed by

$$f^*(A) = \underset{i}{Max}\left(\frac{L_{\overline{X_i}}(A)}{Cap(X_i, \overline{X_i})}\right) \qquad \text{for } i = 1, 2, \ldots, l,$$

where $(X_1, \overline{X_1}), (X_2, \overline{X_2}), \ldots, (X_l, \overline{X_l})$ are possible critical cuts.

For a partial allocation A, both functions f and f^* underestimate the cost, albeit f^* is more accurate than f. We use f instead of f^* to improve the efficiency of the algorithm; the function f^* is used only when a complete allocation is encountered. This function f^* is applied to find the critical cut and compute the data transfer time, i.e., the cost function $COST(A)$.

OFA starts by executing the procedure Find_Possible_Critical_Cuts to locate primary cuts (for the computation of f) and generate possible critical cuts (for the computation of f^*). The search tree is then expanded by recursively calling procedure Sub_OFA. Initially, no file is assigned. The root node is thus denoted as r:(-,-,...,-). (We use a vector notation to present file allocation: For example (3,-,1) means that F_1 is assigned to site 3, F_2 has not yet been assigned, and F_3 is assigned to site 1.)

While expanding on partial allocations the algorithm uses the following rules:

- Larger files are assigned prior to smaller files.
- Among all nodes (or assignment) at the same level in the search tree, the one with the smallest f value is expanded first.
- If two file assignments have the same f value, the one that allocates the file to the ancestor node (or computer sites) in the cut-tree is expanded prior to the assignment that allocates the same file to the descendant node.

```
ALGORITHM Optimal_File_Allocation
    /* f*_op, A_op, (Xop, Xop-bar) are global (or static) variables. */
    f*_OP = ∞
    Execute PROCEDURE Find_Possible_Critical_Cuts to generate
        primary cuts and possible critical cuts.
    root node r = (-,-,...,-).
    CALL Sub_OFA(r)
    Return A_OP
END_ALGORITHM

PROCEDURE Sub_OFA(v)
    IF the associative allocation of v, denoted by A_v, is a complete
                    allocation THEN
        Find critical cut (X, X-bar) and calculate f*(A_v)
        IF f* = f*(A_v) < f*_OP THEN
            A_OP = A_v; f*_OP = f*; (Xop, Xop-bar) = (X, X-bar)
        END_IF
        Return
    END_IF
    Expand node v by assigning the next file to each of the
        processing nodes without violating the constraints to
        generate all successive nodes of node v
    Determine the expanding order of all successive nodes.
    FOR each of all the successive nodes, say u
        IF f(A_u) < f*_OP AND L_{X_OP-bar}(A_u) < L_{X_OP-bar}(A_OP) THEN
            CALL Sub_OFA(u)
        END_IF
    END_FOR
END_PROCEDURE
```

4.2 Example

Consider the network shown in Fig. 1a. The conditions are given as follows. Node 3 is the target node; three data files with sizes $|F_1| = 300$k, $|F_2| = 400$k, $|F_3| = 800$k are to be allocated; disk capacities of node 1 is 600 k, node 2 is 1,200 k, node 3 is 100 k, and node 4 is 900 k. OFA results in the following trace (see Fig. 4).

- Initially, v_0 is the root.
- Expand v_0 by allocating file F_3 to get v_1 and v_2.
- Expand v_1 (with $f = 0.7$) by allocating file F_2 to get v_3, v_4, and v_5.
- Expand v_3 (with $f = 0.7$) by allocating file F_1 to get v_6 and v_7.
- Since node 2 is the ancestor of node 4 in the cut-tree, expand v_6 and obtain v_8. Compute $f^*(= 0.96)$ and obtain

# of data files	LPA		OFA	
	# of valid allocations	Execution time	# of states	Execution time
1	20	0.2s	22	7ms
2	400	4s	76	8ms
3	7890	39s	832	14ms
4	159600	1596s	16261	150ms
5	3228480	32ks	346672	28s

Fig. 5. Performance comparison between LPA and OFA.

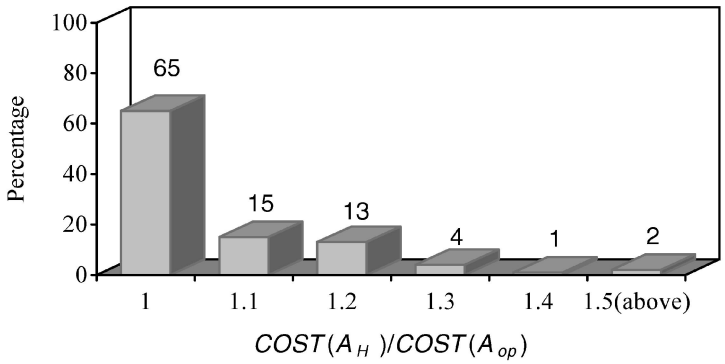

Fig. 6. Performance of HFA.

critical cut $(\{1,3\},\{2,4\})$. So, $f^*_{OP} = 0.96$; $A_{OP} = (2,1,2)$; $(Xop, \overline{Xop}) = (1,3,2,4)$. Return to the higher level.

- Since the f value of v_7 is not less than f^*_{OP}, return to the higher level.
- Since the f values of v_4 and v_5 are not less than f^*_{OP}, return to the higher level.
- Expand v_2 (with $f = 0.94$) by allocating file F_2 to get v_9 and v_{10}.
- Expand v_9 (with $f = 0.94$) by allocating file F_1 and get v_{11}.
- Since the f value of v_{11} is not less than f^*_{OP}, return to the higher level.
- Since the f value of v_{10} is not less than f^*_{OP}, return to the higher level.
- Return to the main program and the program terminates.

4.3 Performance

Here, the execution time of OFA is compared with that of the linear programming approach (LPA). LPA is implemented by enumerating all valid allocations and, then, solving the corresponding linear programming problems to determine the optimal allocation. The simplex method [16] is used to solve linear programming problems in LPA. In this work, the LPA and OFA programs are coded in C++ and run on a PC (Pentium 100).

Our experiments investigate the performance of OFA by varying the number of files to be allocated on a network with 21 nodes and 26 links. File sizes are randomly generated, ranging from 100 k to 1,500 k. Each node (except the target node) can be allocated with, at most, two data files. Allocating data files on target node in disallowed.

Fig. 5 summarizes the performance comparison between the two algorithms. OFA performs much better than LPA. When the number of data files exceeds 3, OFA is about 104 times faster than LPA.

5 Heuristic File Allocation Algorithm (HFA)

OFA may fail to yield an optimal solution in a reasonable amount of time if the number of data files is large. We modify OFA by letting the searching stops as soon as the first complete allocation is obtained. The modified algorithm, known as HFA, has a polynomial-time complexity. The analysis of time complexity of HFA is as follows: In the phase of constructing the cut-tree, $N-1$ maximum flow computations must be performed [8]. Since an $O(N^3)$ algorithm is available for the maximum flow computation [13], the cut-tree construction has time complexity $O(N^4)$. In the second phase of file allocation, the tree-expanding process has a complexity of $O(MN)$, where M is the number of data files to be allocated. Since M is often small than N^3, the complexity of HPA is $O(N^4)$.

To compare HFA and OFA, we consider the test data used in Section 4. For example, with four data files, OFA visits 16,261 nodes in the search tree, while HFA visits only 81 nodes. For the case of five data files, the number of nodes visited by OFA rapidly increases to 346,672, while HFA visits only 101 nodes. The performance of HFA for the test cases described in Section 4 is shown in Fig. 6 (as histograms). The X-axis in Fig. 6 is the ratio $COST(A_H)/COST(A_{OP})$, where A_H denotes the assignment obtained by HFA and A_{OP} represents the optimal assignment. The Y-axis shows the percentage of test cases for the various values on X-axis.

6 Conclusion

In this paper, we described a multipath routing model for the allocation of files in a distributed system using the virtual circuit scheme. Our research addressed efficient techniques for solving the file allocation problem (FAP) which attempts to minimize data transfer time based on the multipath routing. Two algorithms (OFA and HFA) using the branch & bound technique are developed to obtain an optimal and a suboptimal allocations.

The evaluation functions used in branch & bound are based on the critical cut concept. The experimental results suggest that the critical cut approach outperforms the linear programming approach in finding an optimal file allocation and routing of the data. The results also suggest that the heuristic file allocation algorithm (HFA) is a good approximation for the optimal FAP.

Our data transmission model is appropriate for an application requiring massive data transfer from multiple files, such as multimedia database retrieval. It is also applicable to different types of applications that can be modeled as multiple-source-single-sink flow problems.

ACKNOWLEDGMENTS

The work is partially supported by the National Science Council in Taiwan under contract No. NSC872213E009094.

REFERENCES

[1] P.S. Chen and J. Akoka, "Optimal Design of Distributed Information Systems," *IEEE Trans. Computers,* vol. 29, no. 12, pp. 1,068-1,080, Dec. 1980.
[2] R.S. Chen, D.J. Chen, and Y.S. Yeh, "Reliability Optimization of Distributed Computing Systems Subject to Capability Constraints," *Int'l J. Computers & Math. with Application,* vol 29, no. 4, pp. 93-99, 1995.
[3] W. Chu, "Optimal File Allocation in a Computer Network," *Computer-Comm. Systems,* pp. 82-94, Englewood Cliffs, N.J.: Prentice Hall, 1973.
[4] L.W. Dowdy and D.V. Foster, "Comparative Models of File Assignment Problem," *ACM Computer Surveys,* vol. 14, no. 2, pp. 287-313, June 1982.
[5] A.E. El-Abd, "Modeling Resources Allocation and Performance Measures in Distributed Computer Networks," *Proc. IEEE Singapore Int'l Conf. Networks/Int'l Conf. Information Eng.,* pp. 581-586, July 1995.
[6] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia: A Discussion of the Telnet Approach," *Computer Networks and ISDN Systems,* pp. 1,267-1,280, July 1994.
[7] D.V. Foster, L.W. Dowdy, and J.E. Ames, "File Assignment in a Computer Network," *Computer Network,* vol. 5, pp. 341-349, Sept. 1981.
[8] T.C. Hu, *Combinatorial Algorithms,* pp. 60-83. Menlo Park, Calif.: Addison-Wesley, 1986.
[9] B. Jabbari, "A Bandwidth Allocation Technique for High Speed Networks," *Proc. IEEE GLOBECOM '90,* vol. 1, pp. 355-359, Dec. 1990.
[10] J.F. Kurose and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems," *IEEE Trans. Computers,* vol. 38, no. 5, pp. 705-717, May 1989.
[11] A. Lazar and C. Pacifici, "Control of Resources in Broadband Networks with Quality of Service Guarantees," *IEEE Comm.,* pp 66-73, Oct. 1991.
[12] S. Mahmoud and J.S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *ACM Trans. Database Systems,* vol. 1, no. 1, pp. 66-78, Mar. 1976.
[13] V. Malhotra, M. Pramodh Kumar, and S.N. Maheshwari, "An $O(|v|^3)$ Algorithm for Finding Maximum Flows in Networks," *Information Processing Letters,* vol. 7, no. 6, pp. 277-278, 1978.
[14] L. Morgan and K.D. Levin, "Optimal Program and Data Locations in Computer Networks," *Comm. ACM,* vol. 20, no. 5, pp. 315-322, May 1977.
[15] R. Narsinhan, "Database Allocation in a Distributed Environment: Incorporating Concurrency Control & Queuing Costs," *Management Science,* vol. 40, no. 8, Aug. 1994.
[16] M.J. Panik, *Linear Programming: Mathematics, Theory and Algorithms*. Kluwer Academic, 1996.
[17] R.M. Pathak, A. Kumar, and Y.P. Gupta, "Reliability Oriented Allocation of Files on Distributed Systems," *Proc. IEEE Symp. Parallel and Distributed Processing,* pp. 886-893, Dec. 1991.
[18] S. Ram and R.E. Marsten, "A Model for Database Allocation Incorporating a Concurrency Control Mechanism," *IEEE Trans. Knowledge and Data Eng.,* vol. 3, pp. 389-395, Sept. 1991.
[19] V. Ramamoorthy and K.M. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems," *J. ACM,* vol. 17, no. 3, pp. 426-445, July 1970.
[20] P.H. Winston, *Artificial Intelligence,* pp. 81-100. Addison-Wesley, 1992.