ELSEVIER

Information Sciences 120 (1999) 143–157

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

An efficient multipath routing for distributed computing systems with data replication

D.J. Chen *, P.Y. Chang

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsin Chu 30050, Taiwan, ROC

Received 3 February 1999; received in revised form 4 May 1999; accepted 2 August 1999

Abstract

In distributed computing environments, executing a program often requires the access of remote data files. An efficient data routing scheme is thus important for time-critical applications. To ensure a prior desired communication quality, we present a connection-oriented routing scheme, the multipath routing, which allows multiple routes to be established between the source and the destination. Based on the multipath routing scheme, the problem of finding a collection of routing paths for an application to minimize its data transmission time is addressed. Such a problem becomes a complex combinatorial one when the application accesses multiple replicated data sources. Since finding an optimal solution is computationally infeasible in practice, we thus propose a heuristic method to get a sub-optimal solution. © 1999 Elsevier Science Inc. All rights reserved.

Keywords: Virtual circuit; Multipath routing; Flow; Cut

1. Introduction

This paper presents a routing algorithm for time critical applications which need to retrieve remote high-bandwidth data file(s) in a LAN-based distributed computing system (DCS). A connection-oriented data forwarding technique,

* Corresponding author. Fax: +886-03-572-4176.
E-mail address: djchen@csie.nctu.edu.tw (D.J. Chen)

PII: S0020-0255(99)00056-0

virtual circuit [4,6,7], which enables bandwidth to be reserved through the lifetime of a connection, is used in the network. Since the inherent link delay (including propagation delay and transmission delay) is insignificant for bulk data transfer in a high-speed LAN, the design objective of the routing algorithm is to maximize data transfer rate (or throughput). For a source-to-destination traffic session, throughput can frequently be improved by splitting the traffic over several paths. The technique of using multiple paths between a source–destination pair is called multipath routing.

Based on the multipath routing scheme, the problem of optimal routing can be described as a multicommodity flow problem if the number of paths in a source–destination session is unlimited and the inherent link delay and control overhead are negligible. Linear programming is hence a feasible technique to solve it. However, since in a DCS data files are often replicated to improve system (or program) reliability, the program may acquire data from any replica [1,3]. This problem thus becomes a complex combinatorial one. An exhaustive approach could of course find the optimal solution, but it pays for high computation price. Our previous work has developed an alternate, the critical-cut algorithm, to solve the optimal routing problem [2]. For most experiment cases, the algorithm yields short execution times. Occasionally, it still takes exponential time. This paper thus proposes a heuristic routing algorithm to obtain acceptable routing paths. Short and stable execution time makes the proposed algorithm be applicable to the existent systems.

Assumptions: 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 for communication between a source and a destination, and multipath routing is supported in the virtual circuit schemes. Intermediate nodes do not buffer packets, but simply send all received packets immediately. Inherent link delay is negligible. A given file cannot be split and distributed to multiple nodes. A communication path cannot have loops.

2. Critical-cut algorithm

This section presents the critical-cut algorithm to find the optimal routing. Proofs of theorems behind the algorithm can be referred in [2].

Notations

source	the node which holds data files
target	the node which issues request for data files
cut	set of edges such that the removal of the edges separates a connected graph into two disconnected subgraphs

$(X,\overline{X})$	the cut separating the nodes in set X from the other nodes (i.e., the nodes in $\overline{X}$). Note that in presenting a cut by the notation $(X,\overline{X})$, target is always placed in X throughout this paper
cap $(X,\overline{X})$	capacity of the cut $(X,\overline{X})$, i.e. sum of the capacities of the edges in set $(X,\overline{X})$
L_V	the number of bytes from the data files in V, where V denotes a set of nodes

2.1. Cut-trees

The cut-tree theorem was proposed in order to effectively find the maximum flow values between all pairs of nodes in a network [5]. For a network with n nodes, to find the maximum flow values between all pairs of nodes, one must compute $n*(n-1)/2$ such maximum flow values. If the network topology is a tree, the maximum flow values between any two nodes can be easily determined by finding the minimum edge capacity in the unique path connecting the two nodes. Two networks are said to be *flow-equivalent* if the maximum flow values between all pairs of nodes are the same. The construction of a flow-equivalent tree from a general graph G is presented in [5]. Therein, a tree that is flow-equivalent to G is called the *cut-tree* of G because each link of the tree represents a minimum cut of G. A cut-tree of G with n nodes shows the $n-1$ minimum cuts of G that do not cross each other. Consider the example in Fig. 1. Fig. 1(a) illustrates an original network G. The numbers associated with the links denote the link capacities. Fig. 1(b) depicts the cut-tree of G generated by the cut-tree algorithm. Fig. 1(c) illustrates the cuts corresponding to the links in the cut-tree. These cuts are referred to herein as *partitioned cuts* since they do not cross each other.

2.2. Critical cut

When data are transferred with maximum rate, the cuts which are saturated are referred to as the *critical cuts*. Critical cuts can be found by enumerating the

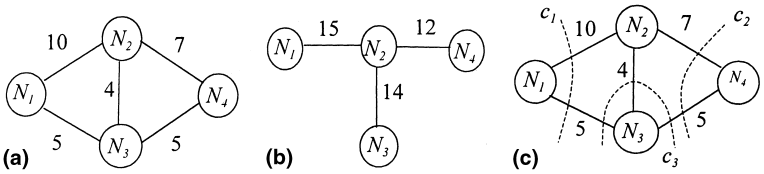


Fig. 1. The relationship between a graph and a cut-tree. (a) The original network G. (b) The cut-tree of G. (c) The partitioned cuts.

cuts that separate sources from the target. Consider a DCS in Fig. 2, where a program on node N_1 needs to retrieve data files F_1, F_2, and F_3. Assume that F_1 on N_3, F_2 on N_4, and F_3 on N_2 are selected to be transmitted. All the possible cuts that separate sources from the target are shown as the dotted lines in Fig. 2. The corresponding parameters in determining the critical cut are listed in Table 1. The term $L_{\overline{X}}/cap(X,\overline{X})$ denotes the minimum time required to transmit data through the cut $(X,\overline{X})$. The overall data transfer time is thus bound by cut c_2 (critical cut). Data are transferred through cut c_2 requires 1.13 s.

In general, enumerating all the cuts as described above to find the critical cut is prohibitive for a large network. Therefore, the number of cuts to be explored must be reduced. The following section describes how to achieve such a reduction.

2.3. Cut reduction

Some terminology needs to be defined before presenting the said cut reduction.

Definition 1. Let $c_i = (X,\overline{X})$ and $c_j = (Y,\overline{Y})$ be two non-crossed cuts. 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 is a *descendant* of c_i. If $\overline{X} \cap \overline{Y} = \emptyset$, c_i and c_j are said to be *brothers* of each other.

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

Definition 2. A partitioned cut c is called the *primary cut* if c has no ancestor cut or the capacity of c is smaller than the capacity of any ancestor cut of c (except c itself).

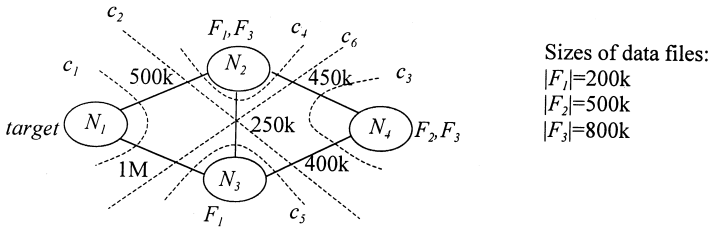


Fig. 2. A simple distributed computing system.

Table 1
Cuts and the corresponding parameters for the example in Fig. 2

Cut	X	$\overline{X}$	$L_{\overline{X}}$	$cap(X,\overline{X})$	$L_{\overline{X}}/cap(X,\overline{X})$
c_1	$\{N_1\}$	$\{N_2,N_3,N_4\}$	$\|F_1\|+\|F_2\|+\|F_3\|=1500$ k	500 k+1 M = 1500 k	1
c_2	$\{N_1,N_3\}$	$\{N_2,N_4\}$	$\|F_2\|+\|F_3\|=1300$ k	500 k+250 k+400 k = 1150 k	1.13
c_3	$\{N_1,N_2,N_3\}$	$\{N_4\}$	$\|F_2\|=500$ k	450 k+400 k = 850 k	0.59
c_4	$\{N_1,N_3,N_4\}$	$\{N_2\}$	$\|F_3\|=800$ k	500 k+250 k+450 k = 1200 k	0.67
c_5	$\{N_1,N_2,N_4\}$	$\{N_3\}$	$\|F_1\|=200$ k	1 M+250 k+400 k = 1650 k	0.12
c_6	$\{N_1,N_2\}$	$\{M_3,N_4\}$	$\|F_1\|+\|F_2\|=700$ k	1 M+250 k+450 k = 1700 k	0.41

Definition 3. Let $c_i \equiv (X, \overline{X})$ and $c_j \equiv (Y, \overline{Y})$ be two non-crossed 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 is an ancestor of c_j, then $c_i \oplus c_j \equiv (X \cup \overline{Y}, \overline{X} \cap Y)$.

Theorem 1. *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 the others.*

The result of Theorem 1 suggests that we may apply exclusive-or operations on primary cuts in order to generated a set of cuts, say S, which contains the critical cut. Procedure *Find-Possible-Critical-Cuts* realizes the process to generate S.

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. In building the cut-tree, only the sources and the target are considered.
Decide primary cuts
Add all primary cuts to set S
Enqueue all primary cuts to Q
REPEAT
$c = \text{dequeue}(Q)$
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: $c \cap c_p = \emptyset$ // 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

2.4. Selection of file–servers

The routing optimization problem becomes a complicated combinatorial problem if multiple replicated copies of data files are dispersed over the network. The following theorem averts a situation which all of the combinations are exhausted to obtain the optimal solution.

Theorem 2. *Let $M1$ and $M2$ be two possible file–server combinations. $M1$ differs from $M2$ only in selecting the copy of file F. $M1$ selects a copy of F from node i; $M2$ selects a copy of F from node j. If node i is the ancestor node of node j in cut-tree, then $TMIN_{M1} \leqslant TMIN_{M2}$ where $TMIN_{M1}$ and $TMIN_{M2}$ denote the data transfer times via the optimal routing paths for $M1$ and $M2$, respectively.*

An efficient method for obtaining the optimal solution is described by the following *Find-Minimum-Transmission-Time* algorithm. In this algorithm, the file–server combination with the smallest "weight" to the root (i.e. the target) is processed first. This strategy stems from Theorem 2, suggesting that the copy nearest to the root in the cut-tree tends to have the shortest transmission time. The *weight* of a file-source mapping is defined as follows.

Definition 4. The *weight* of a file–server combination M is defined as

$$\sum_{N_i \text{ selected by } M} [(d(N_i) \times n(N_i)],$$

where $d(N_i)$ is the number of hops (links traversed) from node N_i to the root in the cut-tree, and $n(N_i)$ the number of the required data files in node N_i for M.

Efficiency of this algorithm follows from the fact that many cuts and file–server combinations can be excluded from consideration. Outputs of the algorithm include the best file–server combination, the critical cut, and the minimum data transmission time. Optimal traffic routing can be easily obtained by executing an additional maximum flow computation. A new graph can be constructed from the original network by adding a super node S^* and connecting S^* to the source nodes with proper edge capacity assignment. Applying the maximum flow algorithm on this new graph, allowS us to obtain the optimal traffic routing.

Algorithm (*Find-Minimum-Transmission-Time*)
// M_{OP} the optimal valid file–server combination.
// $TMIN_{OP}$ the minimum transmission time for M_{OP}.
// CC_{OP} critical cut for M_{OP}.
// $VinXbar_M$ the number of bytes of the requested data in $\overline{X}$ for the valid file–server combination M, where $(X, \overline{X})$is a critical cut found for M.
Execute Procedure *Find-Possible-Critical-Cuts*.
Create the set of valid file–server combinations, say U.
$TMIN_{OP} = \infty, CC_{OP} = \emptyset, M_{OP} = \emptyset$
REPEAT
Select a valid file–server combination with the minimum weight, say *Mi*, from U
Find $TMIN_{Mi}$ and critical cut $(X, \overline{X})$ for *Mi*.

FOR each valid file–server combination Mj in U DO if $VinXbar_{Mj} \geqslant VinXbar_{Mi}$ then remove Mj from U
END_FOR
IF $TMIN_{Mi} < TMIN_{OP}$ THEN
$TMIN_{OP} = TMIN_{Mi},\ CC_{OP} = (X, \overline{X}),\ M_{OP} = Mi$
END_IF
UNTIL $U = \emptyset$
Calculate routing paths for M_{OP}
END_ALGORITHM

2.5. An illustrative example

Consider a DCS in Fig. 2, where a program running on node needs to retrieve data files F_1, F_2 and F_3. Fig. 3 illustrates the cut-tree and the corresponding non-crossed cuts. In this network, there are two copies of F_1 in nodes N_2 and N_3. According to Theorem 2, the copy of F_1 in N_3 is selected to be transmitted since N_3 is the ancestor of N_2. Similarly, there are two copies of F_3 in nodes N_2 and N_4 and the copy in N_2 is selected to be transmitted. We thus obtain an unique file–server combination: selecting F_1 from N_3, F_2 from N_4, and F_3 from N_2. Since all the partitioned cuts (c_1, c_2, and c_3) are primary, the procedure *Find-Possible-Critical-Cuts* begins by adding cuts c_1, c_2, and c_3 to S, the set of possible critical cuts. Table 2 lists the details of this process to generate other possible critical cuts. Since cuts c_1, c_2, and c_3 have ancestor/descendent relationship, no more new possible critical cuts can be generated. The minimum data transfer time can be obtained by

$$TMIN_{OP} = \mathrm{MAX}\left(\frac{|F_1|+|F_2|+|F_3|}{Cap(c_1)}, \frac{|F_2|+|F_3|}{Cap(c_2)}, \frac{|F_2|}{Cap(c_3)}\right)$$
$$= \mathrm{MAX}\left(\frac{1500}{1500}, \frac{1300}{1150}, \frac{500}{850}\right) = \frac{1300}{1150} = 1.13$$

and cut c_2 is the critical cut.

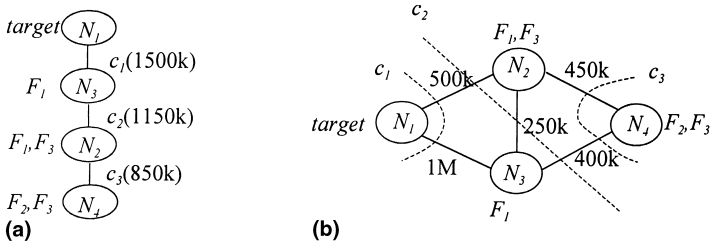


Fig. 3. The cut-tree and the corresponding partitioned cuts for the example in Fig. 2. (a) The cut-tree. (b) Partitioned cuts.

Table 2
The process to generate S, the set of possible critical cuts

Cut	Enter S?	Remark
c_1	Yes	Primary cut
c_2	Yes	Primary cut
c_3	Yes	Primary cut
$c_1 \oplus c_2$	No	c_1 and c_2 has ancestor/descendant
$c_1 \oplus c_3$	No	c_1 and c_3 satisfy ancestor/descendant
$c_2 \oplus c_3$	No	c_2 and c_3 satisfy ancestor/descendant

Traffic routing of this optimal transmission can be obtained by executing the maximum flow algorithm on the graph in Fig. 4. The graph in Fig. 4(a) is constructed by adding a supernode S^* and the pseudoedges between S^* and the source nodes. The capacities of edges between S^* and nodes N_2, N_3, and N_4 are given as 800 k/1.13 = 707, 200 k/1.13 = 176, and 500 k/1.13 = 442 k, respectively. Fig. 4(b) presents the solution of the maximum flow from node S^* to node N_1, where the supernode and pseudoedges have been removed. Multiplying the flow values on the edges by $TMIN_{OP}$ allows us to obtain the volume of data transmitted via the edges.

3. Heuristic routing algorithm

The performance of critical-cut method depends on the cut tree and the distribution of files. For the case such as the example in Section 2.5, the cut tree is a linear array and the algorithm therefore yields very short execution time. However, for some other cases, critical cut method may take exponential time to obtain the optimal routing. In practical environments, a routing algorithm with short and stable execution time is important in real-time applications. We thus propose a heuristic method to decide the routing paths. The main concept of the proposed algorithm is quite simple. Each file is assigned a proper portion

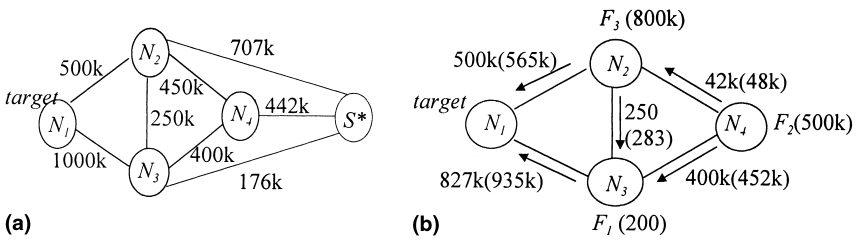


Fig. 4. The way to obtain the traffic flow along each edge. (In (b), values in parenthesis denote the data volume shipped via the edges.) (a) Graph with supernode S^* and pseudoedges. (b) The flow traffic of the optimal solution.

of the total available communication capacity for its transfer. The reserved communication capacity for a data file is proportional to the size of the file. The routing strategy is that all data files are routed simultaneously, and, ideally, will complete their transfers at the same time. This strategy stems from the optimal routing. Compared with the intuitive approach in which data files are routed one after the other, such a simultaneous file transfer can result in higher link utilization and can have shorter transmission time.

However, the total available communication capacity (i.e. the capacity of the critical cut) is not easily determined. The algorithm thus uses the available capacity around the target instead to calculate the appropriate communication capacity for each file's transfer. For the DCS in Fig. 2, the routing algorithm will initially assign the capacities of 200 ($= 1500 \times 200/1500$) kbytes/s for F_1's transfer, 500 ($= 1500 \times 500/1500$) kbytes/s for F_2's transfer, and 800 ($= 1500 \times 800/1500$) kbytes/s for F_3's transfer. The algorithm then operates in a breadth-first-search (BFS) order, with the target as the root of the graph, to visit the nodes and try to find the paths with the assigned communication capacities to transmit the data files (if any). Unfortunately, it is typically not the case that all of the files can be transmitted with the initially assigned capacities. Some files that are located on nodes far from the target may fail to get their deserved communication capacities. These files will then be transmitted at a lower speed or be pending until the transmission of some other file is finished and the occupied channels are released. The released capacity will then be allocated to transmit those files that have not been completely transferred. This process repeats until all the data files are entirely delivered. The algorithm is shown below.

Algorithm (*Heuristic multipath routing*)

// FS set of data files. Initially, all data files are in FS.
// $FS1$ set of data files. Initially, $FS1$ is empty.
// NS_d set of nodes with distance d, where "distance" is the minimum number of links between the node and the target.
// *Maxdist* the longest distance in the network.
// $Cap(N_i, N_k)$ available capacity of the link connecting nodes N_i and N_k.
Complete = FALSE
WHILE NOT Complete
  Compute the deserved capacity for each data file in FS.
  FOR $d = 1$ to *Maxdist* DO
    WHILE $NS_d \neq \emptyset$ DO
      Select a node N_i which have the largest value of $\sum_{N_K \in NS_{d-1}} Cap(N_i, N_k)$, from set NS_d.
      WHILE node N_i is in NS_d DO
      IF there are no files in FS can be found in N_i or there are no paths connecting N_i and target THEN

Remove node N_i from set NS_d.
ELSE
Transmit file F_j (where F_j is in FS and a copy of F_j is found in node N_i) via established virtual circuit(s) with the capacity as computed for F_j's transfer, or with the current maximum communication capacity if F_j's deserved communication capacity is not satisfied (in such case, add F_j to set $FS1$).
Remove file F_j from set FS.
END_IF
END_WHILE
END_WHILE
END_FOR
IF file F_j's transmission-completed signal is received THEN
Release the communication channels allocated to F_j.
$FS = FS \cup FS1 - \{F_j\},\ FS1 = \emptyset$
IF $FS = \emptyset$ THEN
Complete = TRUE
ELSE
Send signals to pause the transmission of files in FS and release the allocated channels.
Adjust the sizes of files still remain in FS to be the lengths of untransmitted portions.
END_IF
END_IF
END_WHILE
END_ALGORITHM

As an example, consider the DCS in Fig. 2, where $FS = \{F_1, F_2, F_3\}$, $NS_0 = \{N_1\}, NS_1 = \{N_2, N_3\}$, and $NS_2 = \{N_4\}$ (see Fig. 5(a)). According to file sizes, the reserved capacities for F_1, F_2, and F_3 are 200, 500, and 800 kbytes/s, respectively. The first file transfer will start at a node selected from NS_1 (if a copy of any required data files can be found in that node). In this example, N_3 is selected since $\sum_{N_K \in NS_0} Cap(N_3, N_k) = Cap(N_3, N_1) = 1$ Mbytes/s, which is greater than $\sum_{N_K \in NS_0} Cap(N_2, N_k) = Cap(N_2, N_1) = 500$ kbytes/s. F_1 in N_3 is then transmitted at the rate of 200 kbytes/s along path N_3–N_1, and the estimated F_1's transfer time is 200 k/200 k = 1 s. FS becomes $\{F_2, F_3\}$. Since there is not any other files in FS can be found in N_3, N_3 is removed from NS_1 (see Fig. 5(b)).

Node N_2 is the next one to be explored. File F_3 in N_2 will be transmitted. The virtual circuits with 800 kbytes/s (for example, 500 kbytes/s along path N_2–N_1, 250 kbytes/s along path N_2–N_3–N_1, and 50 kbytes/s along path N_2–N_4–N_3–N_1) are allocated for F_3's transfer. The estimated transfer time is 800 k/800 k = 1 s. FS becomes $\{F_2\}$. Likewise, no other copies of files in FS

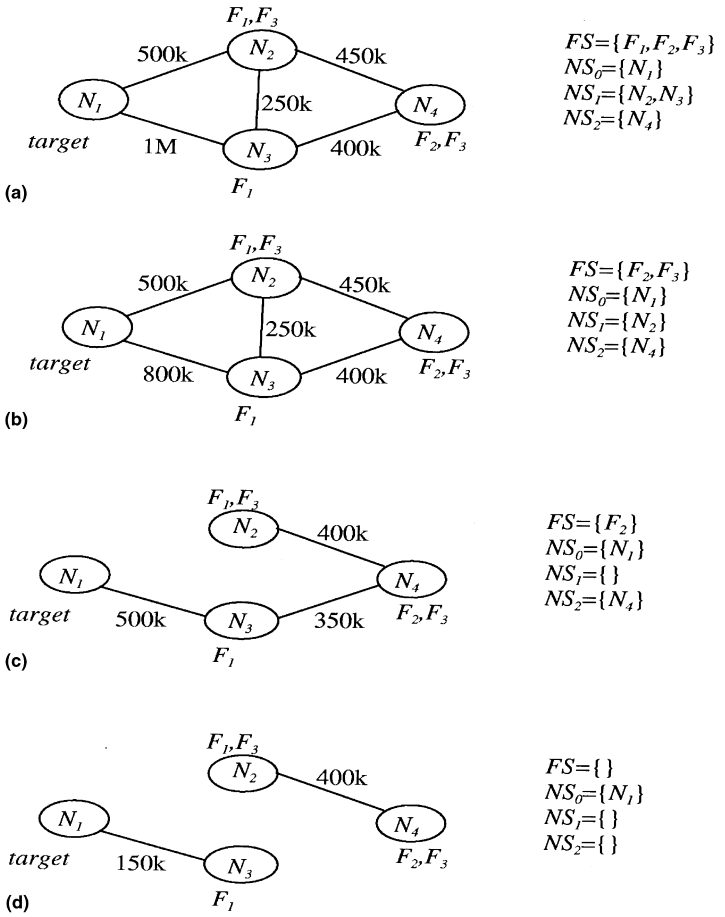


Fig. 5. The statuses for the example in Section 4 by employing *Heuristic multipath routing* algorithm. (a) The initial status. (b) After establishing path N_3–N_1 (200 kbytes/s) for F_1's transfer. (c) After establishing paths N_2–N_1 (500 kbytes/s), N_2–N_3–N_1 (250 kbytes/s), and N_2–N_4–N_3–N_1 (50 kbytes/s) for F_3's transfer. (d) After establishing path N_4–N_3–N_1 (350 kbytes/s) for F_2's transfer.

can be found in N_2, N_2 is removed from NS_1 and thus NS_1 becomes empty (see Fig. 5(c)).

The algorithm will next explore the nodes in set NS_2. Since N_4 is the only node in NS_2, F_2 in N_4 will be transmitted. The algorithm tries to find path(s) with the overall capacity of 500 kbytes/s to transmit F_2. However, only a path (N_4–N_3–N_1) with 350 kbytes/s can be found. F_2 is thus transmitted via such a path. The algorithm then adds F_2 to set $FS1$, and removes F_2 from FS. Fig. 5(d) shows the remaining available capacities after allocating capacities for F_2's transfer.

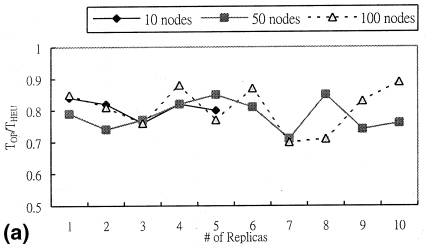
After F_1 and F_3 have completed their file transfers (i.e. 1 s later since their transmissions begin), Set FS is updated as $FS \cup FS1 - \{F_1, F_3\} = \{F_2\}$ and, in addition, size of F_2 is adjusted to be 150 (= 500 k − 350 k) kbytes. The communication channels allocated to F_1, F_2, and F_3 are released. Sets NS_0, NS_1, and NS_2 are reinitialized and, by assuming that the current available link capacities are the same as shown in Fig. 5(a), the reserved communication capacity for F_2 is recomputed as 1500 kbytes/s.

Again, nodes are visited in the sequence given by the algorithm, and finally the untransmitted part of F_2 in N_4 will be transmitted. Since the path(s) with the overall capacity of 1500 kbytes/s are not found, The remaining 150 kbytes of F_2 is transmitted by the paths, N_4–N_2–N_1 (450 kbytes/s) and N_4–N_3–N_1 (400 kbytes/s). The estimated data transfer time for the 150 kbytes is 150 k/(400 k + 450 k) = 0.18 s. When F_2 is completely transferred, FS becomes empty and the routing process stops. Thus the total data transfer time, without considering the control signal processing time and the algorithm execution time, is $1 + 0.18 = 1.18$ s.

4. Simulation results

To evaluate the performance of the proposed heuristic routing algorithm, we compared the routes found by the heuristic algorithm with the optimal routing paths. The optimal routing paths were obtained by the critical-cut method as described in Section 2. Many factors such as the network topology, available link capacities, the number of replicated copies of each data file, and the dispersal of the program and the data files in the sense will influence the performance. In order to fairly evaluate the proposed algorithm, the simulation programs were tested on various scales of randomly generated graphs, with an average node degree in the range of 2.5–3.0. Each link's available capacity and the size of each data file were randomly generated as well.

We use the ratio of T_{OP} to T_{HEU} to measure the efficiency of the routing paths found by the heuristic routing algorithm. Fig. 6(a) shows that all T_{OP}/T_{HEU} values fall into the range between 0.70 and 0.89, no matter what scale of the network is or how many copies each data file has. The relationship of the number of the replicas versus the algorithm execution time is shown in Fig. 6(b)–(d). Obviously, the execution time of the critical-cut algorithm increased with the growth of the number of replicas. On the other hand, the algorithm execution time of the heuristic approach was not affected by such factor. Fig. 6(e) depicts that both algorithms were network size dependent. That is, if the network extends to some scale, both of the routing algorithms are no longer applicable.



T_{OP}: Time to transfer data along the optimal routing paths.
T_{HEU}: Time to transfer data along the paths found by the heuristic routing algorithm.

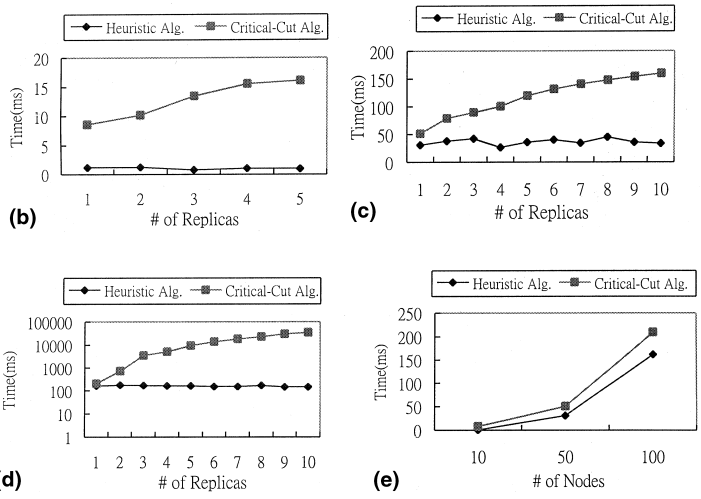


Fig. 6. The comparison of execution time between the heuristic routing algorithm and the critical-cut algorithm. (a) # of replicas vs. T_{OP}/T_{HEU}. (b) # of replicas vs. execution time for 10-nodes graph. (c) # of replicas vs. execution time for 50-nodes graph. (d) # of replicas vs. execution time for 100-nodes graph. (e) # of node vs. execution time (without file replication).

5. Conclusion

In this paper, we propose a heuristic multipath routing algorithm on virtual circuit based DCS. Such a routing scheme is appropriate for high-bandwidth demanded data transfers issued by real-time applications. To evaluate how well the algorithm performs, we made experiments on various networks to monitor the algorithm execution time and the communication capacities of the found routing paths. These observed experimental data are compared with the results obtained by the critical-cut algorithm, which is an efficient optimal routing algorithm proposed in our previous work. Experimental results show that the paths found by the heuristic routing algorithm carry about 70–90% data flow of the optimal routing.

The proposed routing algorithm is a centralized routing. One common problem of the centralized routing is that the time for routing calculation will increase with the network size. Fortunately, this problem may be relieved as more and more powerful computers are developed. We learn from the experiments that the current personal computers execute the algorithm to calculate the routes (for the 100-nodes network) in 0.2 s. For bulk data transfer, such program execution time is acceptable if data transfer time takes seconds.

References

[1] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, MA, 1987.
[2] P.Y. Chang, D.J. Chen, Optimal routing for distributed computing systems with data replication, in: Proceedings of IEEE International Computer Performance and Dependability Symposium, Urbana-Champaign, USA, 1996.
[3] A. El Abbadi, S. Toueg, Maintaining availability in partitioned replicated databases, ACM Transaction on Database Systems 14 (2) (1989) 264–290.
[4] D. Ferrari, A. Banerjea, H. Zhang, Network support for multimedia: A discussion of the telnet approach, Comput. Networks and ISDN System (1994) 1267–1280.
[5] T.C. Hu, Combinatorial Algorithms, Addison-Wesley, Menlo Park, CA, 1986.
[6] B. Jabbari, A bandwidth allocation technique for high speed networks, in: Proceedings of IEEE GLOBECOM'90, vol. 1, San Diego, CA, 1990, pp. 355–359.
[7] A. Lazar, C. Pacifici, Control of resources in broadband networks with quality of service guarantees, IEEE Commun. Mag. (1991) 66–73.