Computer Communications 25 (2002) 1622–1630

computer communications

www.elsevier.com/locate/comcom

A fast algorithm for reliability-oriented task assignment in a distributed system

Chin-Ching Chiu[a,*], Yi-Shiung Yeh[b], Jue-Sam Chou[b]

[a]*Department of Management Information System, Private Takming College, Ney Hwu, Taipei, Taiwan, ROC*
[b]*Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC*

Received 6 March 2001; revised 7 January 2002; accepted 28 January 2002

Abstract

Distributed systems (DS) have become a major trend in computer systems design today because of their high speed and high reliability. Reliability is an important performance parameter in DS design. The distribution of programs and data files can affect the system reliability. Usually, designers add redundant copies of software and/or hardware to increase the system's reliability. The reliability-oriented task assignment problem, which is NP-hard, is to find a task distribution such that the program reliability or system reliability is maximized. In this paper, we developed a reliability-oriented task allocation scheme, based on a heuristic algorithm, for DS to find an approximate solution. The simulation shows that, in most test cases with one copy, the algorithm finds suboptimal solutions efficiently. When the algorithm cannot obtain an optimal solution, the deviation is very small; therefore, this is a desirable approach for solving these problems.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Distributed system reliability; Task assignment; Heuristic algorithm

1. Introduction

Distributed systems (DS) have become increasingly popular in recent years. The advent of VLSI technology and low-cost microprocessors have made distributed computing economically practical in today's computing environment. DS can provide appreciable advantages, including high performance, high reliability, resource sharing, and extensibility [1]. Reliability improvements in DS are possible because of program and data-file redundancy. Reliability evaluations of DS have been widely published [1–8]. To evaluate the reliability of a DS, including a given distribution of programs and data files, it is important to obtain a global reliability measure that describes the degree of system reliability [10–15].

For a given distribution of programs and data files in a DS, distributed program reliability (DPR) [9] is the probability that a given program can be run successfully and will access all of the required files from remote sites despite faults occurring among the processing elements and communication links. The second measure, distributed system reliability (DSR), is defined as the probability that all of the programs in the system can be run successfully. Kumar et al. [9] demonstrated that redundancy in resources, such as computers, programs, and data files can improve the reliability of DS [9]. The study of program and data-file assignment with redundancy considerations is therefore important in improving DSR.

Assume that there are n processing nodes, P programs, F data files and k copies. The total number of possible assignment cases is $n^{k(P+F)}$. Thus, the optimal program and file allocation on the processing nodes are a problem of exponential complexity [15]. This implies that optimum solutions can be found only for small problems. For larger problems, it is necessary to introduce heuristic algorithms that generate near-optimum solutions.

We have presented some algorithms for computing a DS. The first algorithm generates disjoint FSTs by cutting different links and computing the DPR and DSR based on a simple and consistent union operation on the probability space of the FSTs [8]. The second algorithm eliminates the need to search a spanning tree during each subgraph generation. The algorithm reduces both the number of subgraphs generated and the actual execution time is more required for analysis of DPR and DSR [11]. Both algorithms assume that the copies of each program and file set are dynamically dependent upon the system capacity. The third algorithm reduces the total amount of execution time for optimizing the k-node reliability under the given

* Corresponding author. Tel.: +88-62-26585801x393.
E-mail address: chiu@imd.takming.edu.tw (C.C. Chiu).

Nomenclature

$G = (V, E)$	an undirected DS graph, where V denotes a set of processing elements, and E represents a set of communication links
n	the number of nodes in G, $n = \|V\|$
v_i	an ith node represents the ith processing element
$c(v_i)$	the capacity of the ith node
e	the number of links in G, $e = \|E\|$
$e_{i,j}$	an edge represents a communication link between v_i and v_j
$a_{i,j}$	the probability of success of link $e_{i,j}$
$b_{i,j}$	the probability of failure of link $e_{i,j}$
$d(v_i)$	the number of links connected to the node v_i
$w(v_i)$	the weight of the ith node
$w(e_{i,j})$	the weight of the link $e_{i,j}$
F	total number of files in the DS
P	total number of programs in the DS
p_i	distributed program i
f_i	file i
$s(p_i)$	the size of program p_i
$s(f_i)$	the size of data file f_i
pf_i	distributed program or file i
k	the number of copies of pf_i
$\mathrm{AFL}(p_i)$	list of files required for p_i to complete its execution
$\mathrm{APL}(f_i)$	list of programs, which are required f_i to complete their execution
$\mathrm{DPR}(p_i)$	distributed program reliability of p_i
FST	file spanning tree consisting of the root node (processing element that runs the program) and some other nodes, which hold all the files needed for the program held in the root node under consideration
MFST	minimal FST containing no subset file spanning tree
$\mathrm{MFST}(p_i)$	set of minimal file spanning trees associated with program p_i
$s_{need}(v_i)$	the total capacity requires from v_i
PA_i	when a program p_j is assigned to a node v_i, set bit $\mathrm{pa}_{i,j-1}$ of $\mathrm{PA}_i = 1$, otherwise, set bit $\mathrm{pa}_{i,j-1}$ of $\mathrm{PA}_i = 0$
FA_i	when a data file f_j is assigned to a node v_i, set bit $\mathrm{fa}_{i,j-1}$ of $\mathrm{FA}_i = 1$, otherwise, set bit $\mathrm{fa}_{i,j-1}$ of $\mathrm{FA}_i = 0$.
$w(p_i)$	the weight of program i, the value is the total size of files in $\mathrm{AFL}(p_i)$
$w(f_i)$	the weight of data file i, the value is the total size of programs in $\mathrm{APL}(f_i)$

capacity constraint [16]. The size of each program and file set is not discussed separately. Hwang and Tseng proposed k copies of the distributed task assignment (k-DTA) problem. The k-DTA models the assignment of k copies for both distributed programs and their data files to maximize the DSR under some resource constraints. Because the k-DTA problem is NP-hard [13], this study proposed a heuristic algorithm to find an approximate solution. The simulation results show that the exact solution can be obtained in most cases with one copy using the proposed algorithm. When the algorithm fails to obtain an exact solution, the deviation from the exact solution is very small. When the number of copies is greater than one, an approximate solution is obtained, because the number of assigned redundant copies will be constrained on the nodes, whose size is sufficient to allocate the file.

Section 2 defines the task assignment reliability in a DS. Section 3 gives our algorithm and an illustrated example. Section 4 provides the results and a discussion of the problem from Section 2. Our conclusions are presented in Section 5.

2. Computing optimal reliability

To clarify our research objectives, the problem addressed herein is described.

2.1. Definitions

The following definitions are used in this work.

Definition 1. A DS is defined as a system involving cooperation among several loosely coupled computers (processing elements). This system communicates (by links) over a network.

Definition 2. A distributed program is defined as a program for some DS that requires one or more files. For successful distributed program execution, the local host, the processing elements possessing the required files and the interconnecting links must be operational [9].

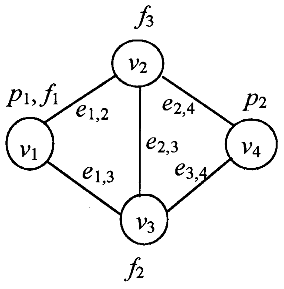


Fig. 1. The DS with four nodes and five links.

Definition 3. A dependent set is defined as a set S of distributed programs and files such that no partition exists that divides S into two disjoint S_1 and S_2, where $S_1 \cup S_2 = S$, and $S_1 \cap S_2 = \varnothing$, such that each program and the files required are within the same subset [13].

Definition 4. The DTA problem is defined finding an assignment for a dependent set under some resource constraints in a DS such that the DS has maximum reliability [13].

Definition 5. A k-DTA problem is defined as determining the assignments for k copies of a dependent set to maximize the DSR under some resource constraints in a DS [13].

2.2. Problem statements

Bi-directional communication channels operate between processing elements. A DS can be modeled using a simple undirected graph. For a DS topology with four nodes and five links, if two programs and three data files should be allocated, the number of different combinations of programs and data files for allocation is 4^5, that is, 1024. For example, the program p_1 requires data files f_1, f_2, and p_2 requires data files f_1, f_2, f_3, for completing execution. Assume that these files are allocated as shown in Fig. 1.

For program p_1, there are three MFSTs, $v_1v_3e_{1,3}$, $v_1v_3e_{1,2}e_{2,3}$, $v_1v_3e_{1,2}e_{2,4}e_{3,4}$. Therefore, $\mathrm{DPR}(p_1) = \Pr(\bigcup_{i=1}^{3} \mathrm{MFST}_i)$. Once all of the minimal file spanning trees have been generated, the next step is to find the probability that at least one MFST is working, which means that all of the edges and vertices included in it are operational. Any terminal reliability evaluation algorithm based on path or cut-set enumeration can be used to obtain the DPR of the program under consideration. The reliability of program p_1 can be computed using a sum of mutually disjoint terms [4].

$$\mathrm{DPR}(p_1) = b_{1,2}a_{1,3} + a_{1,2}a_{1,3}b_{2,3}b_{2,4} + a_{1,2}a_{1,3}b_{2,3}a_{2,4}b_{3,4} + a_{1,2}a_{2,3}a_{2,4}b_{3,4} + a_{1,2}a_{2,3}b_{2,4} + a_{1,2}a_{2,4}a_{3,4}$$

Assume that the probability of each link is 0.9. Then, $\mathrm{DPR}(p_1) = 0.98829$.

In the same way, all of the eight MFSTs for program p_2 are as follows.

$v_1v_2v_3v_4e_{1,2}e_{1,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,4}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,4}e_{3,4}$, $v_1v_2v_3v_4e_{1,2}e_{1,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,3}e_{3,4}$

$$\mathrm{DPR}(p_2) = \Pr\left(\bigcup_{i=1}^{8} \mathrm{MFST}_i\right) = a_{1,2}a_{1,3}b_{2,3}a_{2,4}b_{3,4} + a_{1,2}b_{1,3}a_{2,3}a_{2,4}b_{3,4} + a_{1,2}b_{1,3}b_{2,3}a_{2,4}a_{3,4} + b_{1,2}a_{1,3}a_{2,3}a_{2,4}b_{3,4} + a_{1,2}a_{1,3}a_{2,3}a_{2,4}b_{3,4} + b_{1,2}a_{1,3}b_{2,3}a_{2,4}a_{3,4} + a_{1,2}a_{1,3}b_{2,3}a_{3,4} + b_{1,2}a_{1,3}a_{2,3}a_{3,4} + a_{1,2}a_{2,3}a_{3,4} = 0.97686$$

$$\mathrm{DSR} = \Pr\left(\bigcap_{i=1}^{p} p_i\right) = \Pr\left(\bigcap_{i=1}^{p} \mathrm{MFST}(p_i)\right)$$

Therefore, the results for all MFSTs of the intersection of the two programs are as follows.

$v_1v_2v_3v_4e_{1,2}e_{1,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,4}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,4}e_{3,4}$, $v_1v_2v_3v_4e_{1,2}e_{1,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,2}e_{2,3}e_{3,4}$, $v_1v_2v_3v_4e_{1,3}e_{2,3}e_{3,4}$

$$\mathrm{DSR} = \Pr\left(\bigcap_{i=1}^{p} p_i\right) = a_{1,2}a_{1,3}b_{2,3}a_{2,4}b_{3,4} + a_{1,2}b_{1,3}a_{2,3}a_{2,4}b_{3,4} + a_{1,2}b_{1,3}b_{2,3}a_{2,4}a_{3,4} + a_{1,3}a_{2,3}a_{2,4}b_{3,4} + b_{1,2}a_{1,3}b_{2,3}a_{2,4}a_{3,4} + a_{1,2}a_{1,3}b_{2,3}a_{3,4} + a_{1,2}a_{1,3}b_{2,3}a_{3,4} + a_{1,2}a_{2,3}a_{3,4} = 0.97686$$

A reliability-oriented task assignment problem can be characterized as follows.

Given

Topology of a DS.
The reliability of each communication link.
The available memory space for each processing element.
A set of distributed programs.
A set of data files.
The sizes of each distributed program.
The size of each data file.
Files required by each distributed program for execution.

Assumption

Each node is perfectly reliable.
Each link is either in the working (ON) state or failed (OFF) state.

A graph G does not have any self-loops.
The failure of a link is independent of the failure of other links.
Each node on a DS can have only one copy of the data files.

Constraint

The memory space limitation of each processing element.

Goal

Maximize the DSR of the system (or maximize DPR of a given program).

Reliability optimization can be defined as the maximum reliability for computing a given task under some constraints. For a given task, the reliability can be computed as $R_1, R_2, \ldots, R_x$ for x situations, where x may be an astronomical figure. In doing so, the reliability optimization for the task is the maximal reliability in $R_1, R_2, \ldots, R_x$. The heuristic algorithm involves obtaining an approximate solution, which is close to the maximal reliability in $R_1, R_2, \ldots, R_x$. Restated, a task assignment must be found under the given DS such that the DPR of a given program or DSR of the system is adequate.

The main problem can be mathematically stated as follows:

Given

distributed system parameters,
memory capacity of each node,
memory requirement of each program and data file,
number of copies of each program and data file;

$$\text{Object: maximize DSR} = \Pr\left[\bigcap_{i=1}^{P} p_i\right],$$

subject to

$$\left(\sum_{j=1}^{P} s(p_j)\mathrm{PA}_{i,j} + \sum_{j=1}^{F} s(f_j)\mathrm{FA}_{i,j}\right) \le c(v_i), \quad i = 1, \ldots, n,$$

$$\sum_{i=1}^{n} \mathrm{PA}_{i,j} = k, \quad k \text{ is the number of copies of } \mathrm{pf}_i,$$

$$\sum_{i=1}^{n} \mathrm{FA}_{i,j} = k, \; k \text{ is the number of copies of } \mathrm{pf}_i$$

Obviously, this problem requires a large execution time. Herein, we develop an efficient method that allows task assignment optimization in the DS that achieves the desired performance. Owing to its computational advantages, a proposed method may improve the execution time.

3. Heuristic algorithm for reliability-oriented task assignment methodology

Our problem involves program and data file assignments. The network topology is fixed. The size of every node is also fixed. The proposed algorithm generates two node queues according to the weight of the node and link, respectively, constructs the access relation list and assigns each program and data file dependent upon the above two results.

3.1. Development of HROTA

The development of HROTA is described in the following subsections.

3.1.1. Compute the weight of each node

The number of ports at each node (degree of a node) and number of links directly impact the system reliability. Reliability decreases with a decrease in the number of links [5]. For any node, the degree that node affects the number of information paths that can be transferred from other nodes. The node with the higher degree is more likely to have more paths to the destination nodes than those with lower degrees. Therefore, we employed a simple means for computing the node weight, which takes less time and can quickly compute the weight of every node. The following formula is used to compute the weight of node v_i [16].

$$w(v_i) = 1 - \prod_{z=1}^{d(v_j)} b_{j,t_z} \tag{1}$$

```
Subproc NodeWeight($v_i$)
  Let $E_i = \{e_{i,j} | e_{i,j}$ is incident with $v_i\}$.
  Select a link $e_{i,j}$ from $E_i$.
  Let $w(v_i) = a_{i,j}$.
  Let $E_i = E_i - e_{i,j}$.
  Dowhile ($|E_i| > 0$)
    Select a link $e_{i,j}$ from $E_i$.
    Let $w(v_i) = w(v_i) + (1 - w(v_i))a_{i,j}$.
    Let $E_i = E_i - e_{i,j}$.
  End_Dowhile
  Return $w(v_i)$.
end NodeWeight
```

3.1.2. Sort nodes in descending according to their weight

After obtaining each node weight, all of the nodes are sorted according to their weight in descending order and placed into a queue.

```
Subproc NodeDesByNodeWeight( )
  Let queue $Q_1 = \varnothing$.
  Dowhile (there is a node $v_i \in V$ and $v_i \notin Q_1$)
    Let $v_i = \max\{w(v_i) | v_i \in V$ and $v_i \notin Q_1\}$
    Add $v_i$ to queue $Q_1$.
  End_Dowhile
```

```
    Return $Q_1$.
  End NodeDesByNodeWeight
```

3.1.3. Compute the weight of each link

The reliability of a set of two nodes depends on their links and the link reliability. In the network, two nodes may contain many paths between them. The length of a path is between 1 and $n-1$. To reduce the computational time, we considered a path in which the length is not greater than two. The following formula is used to evaluate the weight of link $e_{i,j}$.

$$w(e_{i,j}) = 1 - b_{i,j}\prod_{z=1}^{y_{i,j}}(b_{i,k_z}a_{k_z,j}) \quad (2)$$

where $y_{i,j}$ denotes the number in which the length of a path between v_i and v_j is two. In addition, $y_{i,j}$ is not greater than $n-2$. The weight of $e_{i,j}$ can be computed in one subtraction and $2(y_{i,j}+1)$ multiplication. Thus, in the worst case, when the graph is a complete graph, we can obtain all of the weights for each link in $n(n-1)/2$ subtractions and $n(n-1)(n-2)/2$ multiplication.

```
  Subproc LinkWeight($e_{i,j}$)
    Let $S_i = \{e_{i,k}e_{k,j} | \text{path } e_{i,k}e_{k,j} \text{ exists between } v_i \text{ and } v_j\}$
    Let $w(e_{i,j}) = a_{i,j}$. /* the reliability of a link $e_{i,j}$. */
    Dowhile ($|S_i| > 0$)
      Select a path $e_{i,k}e_{k,j}$ from $S_i$
      Let $w(e_{i,j}) = w(e_{i,j}) + (1 - w(e_{i,j}))a_{i,k}a_{k,j}$.
      Let $S_i = S_i - e_{i,k}e_{k,j}$.
    End_Dowhile
    Return $w(e_{i,j})$
  End LinkWeight
```

3.1.4. Sort nodes in descending according to all of the link weight

After obtaining the link weights, the nodes are sorted in descending order according to the link weights. This method chooses two nodes incident to the link with the greatest weight first. The order nodes are then appended one by one according to their link weight and the link incident to the selected nodes.

```
  Subproc NodeDesByLinkWeight( )
    Let queue $Q_2 = \emptyset$.
    Let $e_{i,j} = \max\{w(e_{i,j}) | e_{i,j} \in E\}$.
    If ($w(v_i) \geq w(v_j)$)
      Add $v_i$, $v_j$ to queue $Q_2$ in order.
    Else
      Add $v_j$, $v_i$ to queue $Q_2$ in order.
    End_If
    Dowhile(there is a node, say $v_i \in V$, $v_i \notin Q_2$)
      $S_1 = \{e_{i,j} | e_{i,j}$ is incident with a node in queue $Q_2$, either $v_i$ or $v_j$ is in $Q_2\}$
      $S_2 = \{v_i | v_i$ is adjacent to a node in queue $Q_2$ and $v_i$ is not in $Q_2\}$
      Let $v_i = \max\{w(e_{i,j}) | e_{i,j} \in S_1 \text{ and } v_i \in S_2\}$
      Add $v_i$ to queue $Q_2$
    End_Dowhile
    Return $Q_2$.
  End NodeDesByLinkWeight
```

3.1.5. Construct APL(f_i) and AFL(p_i)

In most cases, the total required memory size dominates the number of nodes needed. We therefore simply used the total memory size of p_i and its associated files to approximate the number of required nodes. We used the following formula to evaluate the weight of each program and data file.

$$w(p_i) = \sum s(f_i), \text{ where } f_i \in \text{APL}(f_i).$$

$$w(f_i) = \sum s(p_i), \text{ where } p_i \in \text{AFL}(p_i).$$

Assume that $s(p_1) = 2$, $s(p_2) = 3$, $s(f_1) = 2$, $s(f_2) = 3$, $s(f_3) = 3$, AFL$(p_1) = \{f_1, f_2\}$, AFL$(p_2) = \{f_2, f_3\}$, therefore, APL$(f_1) = \{p_1\}$, APL$(f_2) = \{p_1, p_2\}$, APL$(f_3) = \{p_2\}$, $w(p_1) = s(f_1) + s(f_2) = 5$, $w(p_2) = s(f_2) + s(f_3) = 6$, $w(f_1) = s(p_1) = 2$, $w(f_2) = s(p_1) + s(p_2) = 5$, and $w(f_3) = s(p_2) = 3$. The order of program according to weight is p_2, p_1, and the order of file is f_2, f_3 and f_1. We rearrange the APL(f_i) and AFL(p_i) as follows: AFL$(p_1) = \{f_2, f_1\}$, AFL$(p_2) = \{f_2, f_3\}$, APL$(f_1) = \{p_1\}$, APL$(f_2) = \{p_2, p_1\}$, and APL$(f_3) = \{p_2\}$.

3.1.6. Construct AR-list

An AR-list represents that the sequence of programs and data file assignment will be considered. We constructed an AR-list according to each APL(f_i) and AFL(p_i).

For example, according to Section 3.1.5, the AR-list was constructed as $p_2 \to f_2 \to f_3 \to p_1 \to p_2 \to f_1 \to f_2 \to f_3 \to p_1 \to f_1$ if the number of copies is two.

```
  Subproc AccessRelateList(copy)
    Let queue $Q_3 = \emptyset$.
    Let $pf_i = \max\{w(p_i) | i = 1, \ldots, P\}$. /* start at a program for assignment */
    Let type = 0. /* set task type to denote program */
    Let parent = NULL.
    Add ($pf_i$, type) to queue $Q_3$. /* save the information for processing */
    Add ($pf_i$, type, parent) to the tail of AR-list. /* append the record to AR-list */
    Let k_copy[$pf_i$][type] = k_copy[$pf_i$][type] - 1. /* a copy of the task has assigned */
    Dowhile ($Q_3 \neq \emptyset$)
      Let ($pf_i$, type) = delete ($Q_3$, front).
      Let parent = the address of the node of AR-list that include ($pf_i$, type).
      If (type = 0) /* a program */
        Copy AFL($pf_i$) to $S_1$. /* all files required for the
      current program */
      $next\_type = 1$. /* set task type to denote data file */
    Else /* a data file */
      Copy APL($pf_i$) to $S_1$. /* programs, which are required the data file */
      $next\_type = 0$. /* set task type to denote program */
    End_If
    Let ($Q_4 \neq \emptyset$). /* reset the queue $Q_4$ */
    Dowhile ($S_1 \neq \emptyset$)
      $pf_i$ = delete($S_1$, first). /* obtain a task */
      If ($k\_copy[pf_i][type] > 0$) /* the copies of the task is insufficient */
        If (($pf_i$, $next\_type$) is same as its $grand\_parent$)
          Add ($pf_i$, $next\_type$) to queue $Q_4$. /* delay to assign the task */
        Else /* assign the task to the current node */
          Add ($pf_i$, $next\_type$, $parent$) to $AR\text{-}list$.
          Add ($pf_i$, $next\_type$) to queue $Q_3$.
          Let $k\_copy[pf_i][type] = k\_copy[pf_i][type] - 1$
        End_If
      End_If
    End_Dowhile
    If ($Q_4 \neq \emptyset$)
      Let ($pf_i$, $next\_type$) = delete ($Q_4$, front)
      Add ($pf_i$, $next\_type$, $parent$) to $AR\text{-}list$.
      Let $k\_copy[pf_i][type] = k\_copy[pf_i][type] - 1$.
    End_If
  End_Dowhile
  Return $AR\text{-}list$.
End AccessRelateList
```

3.1.7. Allocate each program and data file according to the AR-list and either Q_1 or Q_2

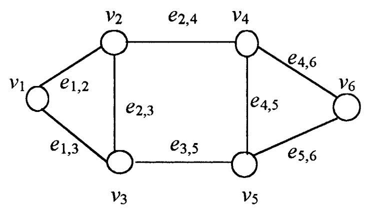
The most reliable assignment for k copies of some program or data file is to assign these copies to k distinct nodes [13]. The total number of programs and file size assigned to a node is at most as large as the capacity of the node.

```
Subproc TaskAssign($AR\text{-}list$, $Q$)
  Let $v_i$ = delete ($Q$, front). /* obtain the weightiest node from queue $Q$ */
  Let $ptr = AR\text{-}list$. /* initialize the pointer $ptr$ as the starting address of $AR\text{-}list$ */
  Let $PA_i = PA_i|(0\times 0001 \leqslant (ptr \rightarrow j-1))$. /* assign $p_j$ to $v_i$ */
  Let $c(v_i) = c(v_i) - s(p_j)$. /* compute the remainder capacity of node $v_i$ */
  Delete the node pointed to by the $ptr$ pointer from the $AR\text{-}list$.
  Let $ptr = AR\text{-}list$. /* move the pointer $ptr$ to next record */
  Dowhile(TRUE) /* traversal each record of $AR\text{-}list$ */
    Let $difference$ = Maxvalue. /* set all of the bits of $difference$ to 1 */
    Dowhile ($ptr! = $NULL) /* use best fit to assign task */
      Let $size = (ptr \rightarrow type = program)?s(p_{ptr\rightarrow j})$: $s(f_{ptr\rightarrow j})$. /* obtain the current task size */
      If ($difference \geq c(v_i)\text{-}size$)
        /* $k$-copies of some program of data file is to assign to distinct nodes */
        If (($ptr \rightarrow type = program$) and ($PA_i$, and $(0\times 0001 \leqslant (ptr \rightarrow j-1))) = 0$)||
            (($ptr \rightarrow type = file$) and ($FA_i$, and $0\times 0001 \leqslant (ptr|j-1))) = 0$)
          Let $candidate\_ptr = ptr$. /* the best record in $AR\text{-}list$ until current */
          Let $difference = c(v_i)\text{-}size$.
        End_If
      End_If
      $ptr = ptr \rightarrow next$. /* examine next record of $AR\text{-}list$. */
    End_Dowhile
    If ($difference$ = Maxvalue) /* $c(v_i)$ is insufficient to assign a program or file */
      Let $v_i$ = delete ($Q$, $front$). /* the next node for assigning task */
    Else
      If ($candidate\_ptr \rightarrow type = program$) /* assign $p_j$ to $v_i$ */
        Let $PA_i = PA_i|0\times 0001 \leqslant (candidate\_ptr \rightarrow j-1))$ /* to assign */
        Let $c(v_i) = c(v_i) - s(p_{candidate\_ptr\rightarrow j})$. /* compute the available capacity of $v_i$ */
      End_If
      If ($candidate\_ptr \rightarrow type = file$) /* assign $f_j$ to $v_i$ */
        Let $FA_i = FA_i|0\times 0001 \leqslant (candidate\_ptr \rightarrow j-1))$ /* to assign */
        Let $c(v_i) = c(v_i) - s(f_{candidate\_ptr\rightarrow j})$. /* compute the available capacity of $v_i$ */
      End_If
      If ($c(v_i) = 0$) /* there is no available capacity in $v_i$ */
        Let $v_i$ = delete ($Q$, $front$). /* next node be consider to assign */
      End_If
      /* the task in the record point by $candidate\_ptr$ pointer has been assigned */
      Delete the node point to by the $candidate\_ptr$ from the $AR\text{-}list$.
    End_If
    ptr = $AR\text{-}list$. /* reset the ptr pointer point to the starting address of $AR\text{-}list$*/
  End_Dowhile
  Return $PA_i$, $FA_i$, for $i = 1, \ldots, n$.
End Task Assign
```

3.2. Complete algorithm of HROTA

The detail steps for HROTA are described as follows:

```
Algorithm HROTA
step 0 /* Initialize DS, and task parameters */
```



$c(v_1)=5$ $c(v_2)=4$ $c(v_3)=6$
$c(v_4)=6$ $c(v_5)=5$ $c(v_6)=5$

$a_{1,2}=0.70$ $a_{1,3}=0.80$ $a_{2,3}=0.90$ $a_{2,4}=0.75$
$a_{3,5}=0.95$ $a_{4,5}=0.85$ $a_{4,6}=0.90$ $a_{5,6}=0.95$

Fig. 2. The DS with six nodes and eight links.

```
    Read DS parameters: $n$, $e$, $a_{i,j}$, $c(v_i)$,
        task parameters: $k$, $P$, $F$, $s(p_i)$, $s(f_i)$, AFL($p_i$).
step 1 /* Compute each node weight using Eq. (1), each link weight using Eq. (2). */
    For each $v_i \in V$ do
        $w(v_i)$ = **NodeWeight**($v_i$). /* call function to evaluate the weight of $v_i$ */
    End_do
    For each $e_{i,j} \in E$ do
        $w(e_{i,j})$ = **LinkWeight**($e_{i,j}$). /* call function to evaluate the weight of $e_{i,j}$ */
    End_do
step 2 /* Generate two queue according to the weight of the node and link, respectively. */
    $Q_1$ = **NodeDesByNodeWeight**( ). /* sort nodes dependent on the node weight */
    $Q_2$ = **NodeDesByLinkWeight**( ). /* sort nodes dependent on the link weight */
step 3 /* Construct APL($f_i$) and AFL($p_i$). */
    Compute the weight of each program and data file, then construct APL($f_i$) and AFL($p_i$).
step 4 /* Construct AR-list */
    **Access Relate List**($k$)
step 5 /* Allocate each program and data file according to the AR-list and either $Q_1$ or $Q_2$. */
    **Task Assign** (AR-list, $Q_1$). /*call function*/
    **Task Assign** (AR-list, $Q_2$). /*call function*/
step 6 /* Compute the DSR and output the best task assignment. */
    Compute the reliability for the final result task assignment indicated at $PA_i$ and $FA_i$ using SYREL [4] and
    Output the task assignment.
End HROTA
```

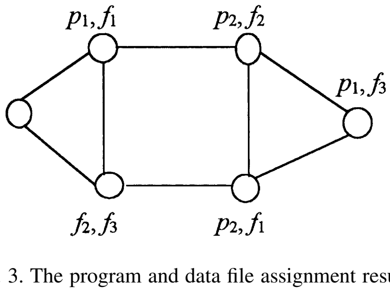


Fig. 3. The program and data file assignment results.

3.3. An illustrate example

The topology of a DS with six nodes and eight links is described as follows. The $c(v_i)$ represents the capacity of node v_i, and $a_{i,j}$ represents the reliability of link $e_{i,j}$ (Fig. 2).

If there are two programs, p_1, p_2, and three data files, f_1, f_2, f_3, with sizes p_1, p_2, f_1, f_2, f_3, is 2, 3, 2, 3, and 3, respectively. The program p_1 needs f_1, f_2, and program p_2 needs f_1, f_2, f_3, for complete execution, e.g. AFL(p_1) is $\{f_1, f_2\}$, AFL(p_2) is $\{f_1, f_2, f_3\}$. Our problem is to find the maximal DS reliability under the allocated programs and files.

In step 1, the weight of each node and link are evaluated using Eqs. (1) and (2), respectively. The weight of v_1, $v_2,\ldots,v_6$ is 0.94, 0.9925, 0.999, 0.9962, 0.9996, 0.995, respectively. The weight of $e_{1,2}$, $e_{1,3},\ldots,e_{5,6}$ is 0.916, 0.926, 0.956, 0.75, 0.95, 0.9783, 0.9808, and 0.9883, respectively.

In step 2, after sorting by the weight of nodes, we obtained $Q_1 = v_5, v_3, v_4, v_6, v_2, v_1$. After sorting by the weight of links, we obtained $Q_2 = v_5, v_6, v_4, v_3, v_2, v_1$.

In step 3, AFL(p_1) = $\{f_2, f_1\}$, AFL(p_2) = $\{f_2, f_1, f_3\}$, APL(f_1) = $\{p_2, p_1\}$, APL(f_2) = $\{p_2, p_1\}$, APL(f_3) = $\{p_2\}$.

In step 4, AR-list $= p_2 \rightarrow f_2 \rightarrow f_1 \rightarrow f_3 \rightarrow p_1 \rightarrow p_2 \rightarrow p_1 \rightarrow f_1 \rightarrow f_2 \rightarrow f_3$.

In step 5, (a) task assignment according to *AR-list* and Q_1, we obtained $PA_1 = 0$, $> PA_2 = 1$, $PA_3 = 0$, $PA_4 = 2$, $PA_5 = 2$, $PA_6 = 1$, and $FA_1 = 0$, $FA_2 = 1$, $FA_3 = 6$, $FA_4 = 2$, $FA_5 = 1$, $FA_6 = 4$. That is, the task assignment is as shown in Fig. 3.

(b) task assignment according to *AR-list* and Q_2, we obtained $PA_1 = 0$, $PA_2 = 1$, $PA_3 = 0$, $PA_4 = 2$, $PA_5 = 2$, $PA_6 = 1$, and $FA_1 = 0$, $FA_2 = 1$, $FA_3 = 6$, $FA_4 = 4$, $FA_5 = 1$, $FA_6 = 2$. That is, the task assignment is as Fig. 3, except f_3 is allocated into v_4 and f_2 is allocated into v_6.

In step 6, generate all of the MFSTs according to the results in step 5. In case (a), we obtained $v_2v_3v_5e_{2,3}e_{3,5}$, $v_2v_3v_4v_5e_{2,4}e_{3,5}e_{4,5}$, $v_1v_2v_3v_5e_{1,2}e_{1,3}e_{2,3}e_{3,5}$, $v_3v_5v_6e_{3,5}e_{5,6}$, $v_2v_3v_4e_{2,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{1,3}e_{2,4}$, $v_2v_4v_6e_{2,4}e_{4,6}$, $v_4v_5v_6e_{4,5}e_{4,6}$, $v_4v_5v_6e_{4,6}e_{5,6}$, $v_4v_5v_6e_{4,5}e_{5,6}$. In case (b), we obtained $v_2v_3v_5e_{2,3}e_{3,5}$, $v_3v_5v_6e_{3,5}e_{5,6}$, $v_1v_2v_3v_5e_{1,2}e_{1,3}e_{2,3}e_{3,5}$, $v_2v_3v_4v_5e_{2,4}e_{3,5}e_{4,5}$, $v_3v_5v_6e_{3,5}e_{5,6}$, $v_2v_3v_4e_{2,3}e_{2,4}$, $v_1v_2v_3v_4e_{1,2}e_{1,3}e_{2,4}$, $v_2v_4v_6e_{2,4}e_{4,6}$, $v_4v_5v_6e_{4,5}e_{4,6}$, $v_4v_5v_6e_{4,5}e_{5,6}$, $v_4v_5v_6e_{4,6}e_{5,6}$. The algorithm computes the DSR using SYREL and Outputs the task assignment. The reliability is 0.9994969. In each case, we use the formula DSR $= \mathrm{pr}(\bigcap_{i=1}^{p} p_i) = \mathrm{pr}(\bigcap_{i=1}^{p} \mathrm{MFST}(p_i))$ to compute its reliability. Therefore, the reliability count is equal to 2.

Table 1
The results obtained using exhaustive method, Hwang and Tseng method and our proposed method for various DS topologies and k-DTAs (n, the number of nodes in G; e, the number of links in G; NRC, the number of reliability computation; Exhaust, the exhaustive method; k, the number of copies of programs and data files)

Size		k-DTA			AFL			Global optimal solution	Exhaust		Hwang and Tseng		Proposed method		
n	e	k	P	F	p_1	p_2	p_3		NRC	Time (s)	Time (s)	Absolute err	Time (s)	NRC	Absolute err
6	8	1	2	3	f_1,f_2	f_1,f_3	–	0.9883041	4032	16	0.11	0.0099509	0.11	1	0
6	8	1	2	4	f_1,f_2,f_3	f_3,f_4	–	0.9883041	18756	145	0.11	0.0099509	0.11	1	0
6	8	1	3	3	f_1,f_2	f_2,f_3	f_1,f_3	0.9883041	13968	55	0.16	0.0207249	0.16	1	0
6	8	1	3	4	f_1,f_3,f_4	f_2,f_3	f_1,f_4	0.9745220	57624	266	0.22	0.0101783	0.22	1	0
6	8	1	3	5	f_1,f_3,f_4	f_2,f_3,f_5	f_1,f_4	0.9745220	210168	1072	0.22	0.0055219	0.22	1	0
6	8	2	2	3	f_1,f_2,f_3	f_1,f_3	–	0.9998719	21312	97	0.16	0.0146482	0.16	1	0.0003515
6	8	2	2	3	f_1,f_2,f_3	f_1,f_2,f_3	–	0.9992175	21312	34	0.22	0.0137254	0.22	1	0.0002513
6	8	2	2	4	f_1,f_2,f_3	f_1,f_2,f_3,f_4	–	0.9984149	11691	19	0.38	0.0003736	0.38	1	0
6	8	2	2	3	f_2,f_3	f_1,f_2	–	0.9998719	21312	104	0.22	0.0002890	0.22	1	0.0007716
6	8	2	2	3	f_2,f_3	f_1,f_3	–	0.9998719	21312	48	0.22	0.0002890	0.22	1	0.0007716
6	8	2	3	3	f_2,f_3	f_1,f_2	$f_1,f_3,$	0.9998698	3699	24	0.27	0.0053454	0.27	1	0.0011843
6	10[a]	2	2	3	f_2,f_3	f_1,f_2	–	0.9999917	10260	113	0.22	0.0127541	0.22	1	0.0000811
6	10[a]	2	3	3	f_1,f_2	f_2,f_3	f_1,f_3	0.9999951	272157	3433	0.44	0.0274293	0.44	1	0.0000882
6	10[a]	2	3	4	f_1,f_2	f_1,f_2,f_3	f_1,f_3,f_4[b]	1.0000000	5288498	38122	0.38	0.0291030	0.38	1	0.0000172

[a] Addition two links $e_{2,5}$, $e_{3,4}$ and $a_{2,3} = 0.9$, $a_{3,4} = 0.95$.
[b] $s(f_4) = 1$; $s(p_1) = 2$, $s(p_2) = 3$, $s(p_3) = 3$, $s(f_1) = 2$, $s(f_2) = 3$, $s(f_3) = 3$, $s(f_4) = 2$, $s(f_5) = 2$; if $k = 1$, then $c(v_1) = 5$, $c(v_2) = 4$, $c(v_3) = 6$, $c(v_4) = 6$, $c(v_5) = 5$, $c(v_6) = 5$,; if $k = 2$, then $c(v_1) = 6$, $c(v_2) = 5$, $c(v_3) = 7$, $c(v_4) = 7$, $c(v_5) = 6$, $c(v_6) = 6$.

4. Results and discussion

Table 1 presents the data on the results obtained using three different methods for various DS topologies with different allocated programs and data files. In contrast to the exhaustive method, the number of reliability computations grows rapidly, when the size of the DS topology or the number of programs and data files are increased. The proposed method is constant, which is independent of the size of the DS topology and the number of programs and data files. The deviation is very small, when the proposed method cannot obtain an optimal solution. These data show that the proposed method is more effective than the conventional methods.

In this paper, we proposed a new technique for solving the k-DTA reliability problem. The complexity of the proposed algorithm in steps $0,\dots,6$ is $O(1)$, $O(n^3)$, $O(en)$, $O(P + F)$, $O(k(P + F))$, $O((P + F)^2)$, and $O(m^2)$, where k denotes the number of copies of the programs and files and m represents the number of paths for the set of assigned nodes [4]. Other notations, such as n, ps, P, F, cc and mc, are defined in Nomenclature. Therefore, the complexity of the proposed algorithm is $O(n^3 + (P + F)^2 + m^2)$. Results obtained from our algorithm were compared with those from the exhaustive method and the Hwang and Tseng method [13]. Although the exhaustive method, which has a time complexity of $O(m^2 n^{k(P+F)})$, can yield the optimal solution, it cannot effectively reduce the number of reliability computations and the time complexity. An application occasionally requires an efficient algorithm for computing reliability owing to resource considerations. Under this circumstance, deriving the optimal reliability may not be a promising option. Instead, an efficient algorithm yielding approximate reliability is preferred. The time complexity for the Hwang and Tseng method [13] is $O(n^3 + kn(P + F) + m^2)$, which is slightly quicker than our method, but the deviation from the exact solution is not ideal [13].

In contrast to the computer reliability problem, which is static-oriented, the task assignment problems in the DS are dynamically oriented, because many factors, such as the DS topology, node capacity, link reliability, the size of the programs or files, files requested by each program, copies of programs and files and the number of paths between each node can significantly affect the efficiency of the algorithm [4]. Thus, quantifying the time complexity exactly is extremely difficult. The accuracy and efficiency of the proposed algorithm were verified by implementing simulation programs in C language executed in Pentium III with 128M-DRAM on MS-Windows 98. In our simulation case, the number of reliability computations for the proposed algorithm was constant. The exact solution can be obtained in most cases, when there is only one program and file copy. In almost every case, if the number of copies exceeds one, the DSR is close to one and the redundant file may constrain the remaining node capacity. The proposed method can obtain an approximate solution, in which the average deviation from the exact solution is under 0.001.

5. Conclusion

This paper presented a heuristic algorithm for computing k-DTA reliability problem. The proposed algorithm uses an effective algorithm to select a program and file assignment set that has maximal or nearly maximal system reliability. Our numerical results show that the proposed algorithm may obtain the exact solution in most one copy cases and the computation time is significantly shorter than that needed for the exhaustive method. When the proposed method fails to give an exact solution, the deviation from the exact solution appears very small. The technique presented in this paper might be helpful for readers to understand the relationship between task assignment reliability and DS topology.

References

[1] K.K. Aggarwal, S. Rai, Reliability evaluation in computer communication networks, IEEE Trans. Reliab. R-30 (1981) 32–35.

[2] K.K. Aggarwal, Y.C. Chopra, J.S. Bajwa, Topological layout of links for optimizing the S–T reliability in a computer communication system, Microelectron. Reliab. 22 (3) (1982) 341–345.

[3] A. Satyanarayna, J.N. Hagstrom, New algorithm for reliability analysis of multiterminal networks, IEEE Trans. Reliab. R-30 (1981) 325–333.

[4] S. Hariri, C.S. Raghavendra, SYREL: a symbolic reliability algorithm based on path and cuset methods, IEEE Trans. Comput. C-36 (1987) 1224–1232.

[5] F. Altiparmak, B. Dengiz, A.E. Smith, Reliability optimization of computer communication networks using genetic algorithms, Proc. IEEE Int. Conf. Syst. Man Cybern. 5 (1998) 4676–4680.

[6] D.W. Coit, A.E. Smith, Reliability optimization of series–parallel systems using a genetic algorithm, IEEE Trans. Reliab. R-45 (1996) 254–260.

[7] D. Torrieri, Calculation of node-pair reliability in large networks with unreliable nodes, IEEE Trans. Reliab. R-43 (1994) 375–377.

[8] D.J. Chen, T.H. Huang, Reliability analysis of distributed systems based on a fast reliability algorithm, IEEE Trans. Parallel Distribut. Syst. 3 (1992) 139–154.

[9] V.K.P. Kumar, C.S. Raghavendra, S. Hariri, Distributed program reliability analysis, IEEE Trans. Software Engng SE-12 (1986) 42–50.

[10] A. Kumar, D.P. Agawal, A generalized algorithm for evaluation distributed program reliability, IEEE Trans. Reliab. R-42 (1993) 416–426.

[11] D.J. Chen, R.S. Chen, T.H. Huang, Heuristic approach to generating file spanning trees for reliability analysis of distributed computing systems, J. Comput. Math. Appl. 34 (1997) 115–131.

[12] P. Tom, C.R. Murthy, Algorithms for reliability-oriented module allocation in distributed computing systems, J. Syst. Software 40 (1998) 125–138.

[13] G.J. Hwang, S.S. Tseng, A heuristic task assignment algorithm to maximize reliability of a distributed system, IEEE Trans. Reliab. R-42 (1993) 408–415.

[14] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (1998) 42–51.

[15] A. Kumar, R.M. Pathak, Y.P. Gupta, Genetic algorithm based approach for file allocation on distributed systems, Comput. Ops. Res. 22 (1) (1995) 41–54.

[16] C.C. Chiu, Y.S. Yeh, R.S. Chen, Reduction of the total execution time to achieve the optimal k-node reliability of distributed computing systems using a novel heuristic algorithm, Comput. Commun. 23 (2000) 84–91.

Chin-Ching Chiu
Education: September 1993–October 2000 PhD in Computer Science and Information Engineering, Department of EE and CS, National Chiao-Tung University. September 1988–June 1991 MS in Computer Science, Department of EE and IE, Tamkang University. He received a BS degree in Computer Science from Soochow University.
Professional Background: December 2000—Now Associate Professor, Department of Information Management, Tak-Ming College. September 1991–November 2000 Instructor, Department of Information Management, Tak-Ming College. April 1984–August 1991 Computer Engineer in Data Communication Institute of Directorate General of Telecommunications. October 1982–April 1984 System Analyst of SYSCOM computer company.
Research Interest: Reliability Analysis of Network and Distributed Systems, Genetic Algorithms, DNA computing.

Yi-Shiung Yeh
Education: September 1981–December 1985 PhD in Computer Science, Department of EE and CS, University of Wisconsin-Milwaukee. September 1978–June 1980 MS in Computer Science, Department of EE and CS, University of Wisconsin-Milwaukee.
Professional Background: August 1988—Now Associate Professor, Institute of CS and IE, National Chiao-Tung University. July 1986–August 1988 Assistant Professor, Department of Computer and Information Science, Fordham University. July 1984–December 1984 Doctorate Intern, Johnson Controls, Inc. August 1980–Octpber 1981 System Programmer, System Support Division, Milwaukee County Gov.
Research Interest: Data security and Privacy, Information and Coding Theory, Game Theory, Reliability and Performance.

Jue-Sam Chou
Education: He is currently working towards the PhD degree in Department of CS and IE, National Chiao-Tung University.
Professional Background: September 1991—Now Instructor, Department of Information Management, Ta-Hua College.
Research Interest: Reliability and Performance Evaluation: Networks and Distributed Systems, DNA computing.