Control and Cybernetics

vol. **30** (2001) No. 1

Task allocation algorithms for maximizing reliability of heterogeneous distributed computing systems

by

A. Mahmood

Department of Computer Science, University of Bahrain, Bahrain

Abstract: The rapid progress of microprocessor and communication technologies has made the distributed computing system economically attractive for many computer applications. One of the first problems encountered in the operation of a distributed system is the problem of allocating the tasks among the processing nodes. The task allocation problem is known to be computationally intractable for large task sets. In this paper, we consider the task allocation problem with the goal of maximizing reliability of heterogeneous distributed systems. After presenting a quantitative task allocation model, we present a least-cost branch-and-bound algorithm to find optimal task allocations. We also present two heuristic algorithms to obtain suboptimal allocations for realistic size large problems in a reasonable amount of computational time. Simulation was used to study the performance of the proposed algorithms for a large number of problems. Also, performance of the proposed algorithms has been compared with a well-known heuristics available in the literature.

Keywords: task allocation, distributed computing, reliability, heuristics, branch-and-bound, A* algorithm.

1. Introduction

A single application program often requires many different types of computations that result in different needs for machine capabilities. Heterogeneous computing secures the effective use of the diverse hardware and software components in a heterogeneous suite of machines connected by a high-speed network to meet the varied computational requirements of a given application (Tan and Siegel, 1998). A task or a program to be run on such a system consists of a set of modules, possibly defined by a partition strategy (Bokhari, 1988). Each of the modules comprising a task will execute on one of the processors and communicate with some other modules of the task using the system's interconnection network. In a general purpose distributed system, modules of a task must be assigned to processors in such a way that the system resources will be utilized effectively and certain objective function optimized. This problem, generally known as task allocation problem, has been shown to be NP-complete for more than two processors (Coffman, 1978). Hence, task allocation problem is computationally intractable for all but small systems.

The task allocation problem has been widely investigated by many researchers. Since performance is a dominant concern in most distributed applications, task allocation models with a cost function based on performance measures (such as communication and execution cost or task completion time) have received more attention (Chu, Holloway et al., 1980; Shen and Tsai, 1985; Bowen, Nokolous et al., 1992; Lee and Shin, 1997). These models are based on the assumption that, for most of distributed applications, task completion time is relatively very small as compared to inter-failure rate of communication links and processors; hence, reliability issues have not been explicitly addressed in these models.

On the other hand, very few models have considered reliability issue explicitly because these models are useful in relatively fewer situations. For example, these models are useful for systems with long mission time spread over hours or days (e.g. space flight long-term monitoring systems), systems that require ultra-high reliability (e.g. commercial flight control system) and systems with a large number of processors and modules (Shatz, Wang, and Goto, 1992).

Hariri and Raghavendra (1986) proposed the problem of task allocation for reliability. They achieved high system reliability by introducing multiple copies of modules. However, they did not give an explicit reliability expression in terms of system parameters. Also, this model guarantees an optimal allocation only when the processors and communication links have the same reliability and each processor runs exactly one module. Bannister and Trivedi (1992) presented a task allocation model for fault tolerant systems. In this model, fault tolerance is achieved through the introduction of redundant copies of modules and optimization is achieved by balancing the load over a homogeneous distributed system. This model, however, neither gives an explicit system reliability measure nor considers failure of communication link which, in many systems, is the least reliable factor (Ma, 1982). Shatz, Wang, and Goto (1992) modeled the problem of task allocation for maximizing system reliability and treated both processors and communication links to be unreliable. This model is based on a cost function, which represents the unreliability caused by execution of modules on processors of various reliability and the unreliability caused by the interprocessor communication. They converted the task allocation problem into a state-space search problem and applied the A* algorithm from artificial intelligence theory (Nilsson, 1971) to obtain optimal allocations. Kartik and Murthy (1997) proved that Shatz's algorithm may get trapped into local minima and presented an improved version of Shatz's algorithm which gives optimal solutions in all cases. A task allocation model for fault-tolerant multicomputer systems is presented in Kim, Lee, and Lee (1997). In this model, fault-tolerance is achieved through the introduction of a backup copy for every process in the system and optimization is achieved through load balancing over a multicomputer system. A similar approach has been used in the Paralex system (Davoli et al., 1996), in which fault-tolerance is achieved by introduction of passive replicas and load is balanced dynamically by late binding of primary process.

This paper investigates the task allocation problem; an essential phase of distributed software design (Shatz and Wang, 1987), with the goal to maximize system reliability. The task allocation problem considered in this paper is formally defined as "Given a distributed system of n heterogeneous processors $(p_1, p_2, \ldots, p_n)$ and a task T made up of m modules $(t_1, t_2, \ldots, t_m)$, assign each of the m modules to one of the n processors so that the system reliability is maximized subject to a set of constraints imposed by the application or environment". The main contribution of this paper is constituted by three algorithms. The first algorithm, a least-cost branch-and-bound algorithm, finds the optimal allocation of task to processors. This algorithm is suitable for small size problems because its worst-case time complexity is exponential. The other two algorithms are polynomial time algorithms to find suboptimal solutions and are suitable for large size problems.

The rest of the paper is organized as follows. In the next section, we present a task allocation model. In Section 3, we present our algorithms for optimal and suboptimal task allocation. Section 4 presents simulation results and Section 5 concludes the paper.

2. Task allocation model

2.1. Assumptions

A distributed system is composed of a set of n heterogeneous processors $(p_1, p_2, \ldots, p_n)$ each having local memory and connected with each other through some communication network. Each processor may have different processing speed, memory size and failure rate. The communication links may have different bandwidths and failure rates. We assume that the failure rates of components (i.e. processors and links) are statistically independent and follow a Poisson distribution (i.e. the time of next failure has an exponential distribution). Similar assumptions have been made by Shatz, Wang, and Goto (1992). The network topology is assumed to be cycle free such as star, bus, hierarchical or connected stars. This means that a unique path exists between any two processors. However, the algorithms presented in this paper can easily be extended for other types of topologies.

The task to be assigned consists of m modules $(t_1, t_2, \ldots, t_m)$. A module may communicate with other modules of the task. The intermodule communication cost (IMC) between modules t_i and t_j is denoted by c_{ij} and is measured in number of words or packets exchanged between the modules. If a module t_i assigned to processor p_k communicates with module t_j assigned to processor p_l $(p_k \neq p_l)$ and p_k is not adjacent to p_l then all the links and processors between p_k and p_l take part in the communication. We define system reliability for a 'mission' as the time interval during which we require the system to be reliable. During this time, a module may execute and communicate with some other modules for more than once. We define the Accumulative Execution Time (AET) of a module as the product of the number of times a module executes during the mission and the average time units for each execution of the module. Similarly, inter module communication cost of two modules is the product of number of times they communicate with each other and average number of words transferred during each communication. The notations used in the paper are summarized in Table 1.

M	Number of modules in the task
N	Number of processors in the distributed system
T	Set of modules to be assigned
t_i	ith module of task T
P	The set of processors in the distributed system
p_k	kth processor in P
X	An $m \times n$ binary matrix corresponding to a task allocation
x_{ik}	An element of X; $x_{ij} = 1$ if t_i is assigned to p_k, otherwise $x_{ij} = 0$
e_{ik}	Accumulative execution time for task t_i running on processor p_k
c_{ij}	Intermodule communication cost (IMC) between t_i and t_j during the mission (measured in words or packets)
λ_k	The failure rate of processor p_k
μ_{kb}	The failure rate of link between p_k and p_b
w_{kb}	Transmission rate of the link between p_k and p_b (words or packets per unit of time) and $w_{kb} = w_{bk}$

Table 1. Notations

2.2. Formulation of the task allocation model

The task allocation problem can be represented as a constrained optimization problem (Shatz, Wang, and Goto, 1992). In the optimization problem, an objective function should reflect all the important aspects of system reliability in terms of a number of variables. Since it is difficult to include all the details in a model, it should capture only the critical aspects of system design. The objective function should have a form that involves less real-time information and yet performs coincident evaluation for a given allocation with minimum computations (Mahmood, 1994). Therefore, two important features of an objective function are evaluation accuracy and efficiency.

The accuracy of an objective function depends on two factors. The first factor is whether the form and the parameters of the objective function are proper. That is, if the objective function indicates that an allocation is better than the others, actual measurements should concur. The second factor is the accuracy of the real-time data used in the evaluation. Therefore, while designing an objective function, one should consider whether it is feasible to acquire the required data. Keeping in mind these considerations, we present a task allocation model similar to that of Shatz, Wang, and Goto (1992).

A successful completion of a task requires that the processors be operational during the time that its modules are being executed and each link between terminal processors of a path be operational during the active period of inter module communication. The reliability of the processor p_k in time interval t is $\exp(-\lambda_k t)$, where λ_k is the failure rate of processor p_k. Under the task assignment X, the reliability of p_k, denoted by $R_k(T, X)$, for the execution of the modules assigned to it during the mission is given by

$$R_k(T,X) = \exp\Big(-\lambda_k \sum_{i=1}^{m} x_{ik} e_{ik}\Big), \tag{1}$$

where $x_{ik} = 1$ if module t_i is assigned to processor p_k during the mission, otherwise $x_{ik} = 0$; and e_{ik} is the accumulative execution time of module t_i on processor p_k. Equation (1) gives the total time spent on executing the modules at p_k. Similarly, the reliability of a single-link path L_{kb} between processor p_k and p_b is given by

$$R_{kb}(T,X) = \exp\Big(-\mu_{kb} \sum_{i=1}^{m}\sum_{j=1}^{m} x_{ik}x_{jb}(c_{ij}/w_{kb})\Big), \tag{2}$$

where c_{ij} is intermodule communication cost between modules t_i and t_j and w_{kb} is the transmission rate of link L_{kb}. If the path L_{st} consists of links $l_1, l_2, \dots, l_k$, and μ_i and w_i are failure rate and transmission rate of link l_i $(1 \le i \le k)$, respectively, then we define the term

$$W_{st} = \frac{\mu_1 + \mu_2 + \ldots + \mu_k}{\dfrac{\mu_1}{w_1} + \dfrac{\mu_2}{w_2} + \ldots + \dfrac{\mu_k}{w_k}}. \tag{3}$$

We call the term W_{st} the adjusted transmission rate for the path L_{st} since it reflects the fact that the behavior of the path depends on both the transmission rates and the reliability of all the links in the path. Note that if all the links in a path have the same transmission rate w then the adjusted transmission rate of the path is w. Also, for a single-link path, the transmission rate and adjusted transmission rate are identical, hence we can rewrite (2) as

$$R_{kb}(T,X) = \exp\Big(-\mu_{kb} \sum_{i=1}^{m}\sum_{j=1}^{m} x_{ik}x_{jb}(c_{ij}/W_{kb})\Big). \tag{4}$$

It should be noted here that if a multiple-link path L_{kb} consists of links $l_1, l_2, \ldots$ $\ldots, l_s$ and failure rate of each link in the path follows a Poisson distribution with failure rates $\mu_1, \mu_2, \dots, \mu_s$ respectively, then the failure rate of L_{kb} also follows a Poisson distribution and is given by $\mu_1 + \mu_2 + \ldots \mu_s$.

The reliability of the system during the mission when task T is assigned to it by allocation X, denoted by $R(T, X)$, is defined as a probability that T can run successfully on the system during the mission under task assignment X, and is given by

$$\begin{aligned} &R(T,X)\\ &= \text{Prob}(T \text{ can run successfully under assignment } X \text{ during the mission})\\ &= \Big[\prod_{k=1}^{n} R_k(T,X)\Big]\Big[\prod_{k=1}^{n-1}\prod_{b>k} R_{kb}(T,X)\Big] = \exp(-COST(X)), \end{aligned} \tag{5}$$

where

$$COST(X) = \sum_{k=1}^{n}\sum_{i=1}^{m} \lambda_k x_{ik} e_{ik} + \sum_{k=1}^{n-1}\sum_{b>k}\sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{kb} x_{ik} x_{jb} (c_{ij}/W_{kb}). \tag{6}$$

The first term in (6) gives the unreliability caused by execution of modules on processors of various reliabilities and the second term reflects the unreliability caused by intermodule communication.

From (5) and (6), it is clear that to maximize the reliability of the system $R(T,X)$, we have to minimize $COST(X)$. Therefore, we formally state the task allocation problem as

$$\begin{aligned} &\text{minimize } COST(X)\\ &\qquad = \sum_{k=1}^{n}\sum_{i=1}^{m} \lambda_k x_{ik} e_{ik} + \sum_{k=1}^{n-1}\sum_{b>k}\sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{kb} x_{ik} x_{jb} (c_{ij}/W_{kb}) \end{aligned}$$

subject to

$$\sum_{k=1}^{n} x_{ik} = 1 \text{ for all } 1 \le i \le m$$

$$x_{ik} = 0 \text{ or } 1 \text{ for all } 1 \le i \le m,\ 1 \le k \le n$$

and any other system constraints such as limited processing capacity, limited memory size, bounds for completion time of a module, assigning a task to a particular set of processors, etc.

Various system constraints can be incorporated into the model. For example, two modules t_i and t_j may be required to be executed on the same processor to yield good response time (Lee and Shin, 1997). This can be expressed as $x_{ik} = x_{jk}$ for $1 \le k \le n$. If a task t_i is required to be allocated to a specific processor p_k due to some system consideration, such as availability of some special hardware attached to a processor, it can be specified by letting the parameter $x_{ik} = 1$ (or e_{il} equal to a large number for all $1 \le l \le n$ where $l \ne k$). The real-time requirement can be given by $\sum_{i=1}^{m} x_{ik} e_{ik} < T_k$ where T_k is the required upper bound for processing the modules that reside on the processor p_k. Similarly, the upper limit on the number of modules that can be assigned to a processor p_k can be specified by $\sum_{i=1}^{m} x_{ik} \le U_k$ where U_k is the maximum number of modules that can be allocated to processor p_k.

3. Task allocation algorithms

The task allocation model described in the previous section results in a 0-1 quadratic programming problem. In this section we present three algorithms, one to find optimal task allocations and other two to find suboptimal allocations.

Our first algorithm, based on the idea of Branch-and-Bound, generates a state-space search tree and searches for an optimal solution in the tree. Before explaining the algorithm, we define the following terms. We say that a task assignment X is *incomplete* if not all the modules of a task are assigned under assignment X. Otherwise, X is a *complete assignment.* A state is defined by a list of n sets $S_1, S_2, \ldots, S_n$, where S_k contains the modules assigned to processor p_k. The state-space search tree is built as follows:

1. Arrange the modules in non-decreasing order of C values, where C_i for a module t_i is given by
$$C_i = \min_{k \in P} e_{ik} + \sum_{j=1}^{m} (c_{ij} / \max_{\substack{b in P \\ b \neq k}} w_{kb}). \tag{7}$$
The first term indicates the execution time of module t_i on the fastest processor and the second term reflects the minimum communication time required by module t_i and t_j to exchange the messages among them if they are allocated to two different processors connected by the fastest link. The reason for ordering the modules with respect to this value is that the modules that communicate heavily and require higher processing time should be considered first for allocation. This point is further discussed after Algorithm 1.
2. The root node has a state $(\emptyset, \emptyset, \ldots, \emptyset)$ indicating that none of modules has been assigned.
3. At each subsequent level, nodes are expanded by assigning next module in the sorted list to every processor to generate n nodes of next level. This procedure ends when all the m modules have been assigned. Fig. 1 shows a state-space tree for two modules to be assigned to two processors.

Note that in a state-space search tree all the internal nodes correspond to incomplete assignments and all leaf nodes correspond to complete assignments. Our goal is to find a leaf node in the tree with an assignment X such that $COST(X)$ given by (6) is minimum. This process is often speeded up by using a function to decide the next node to be expanded (Shatz, Wang, and Goto, 1992). At each expansion, the proposed algorithm expands a node with the minimum value of a cost function, f, given by (Shatz, Wang, and Goto, 1992)

$$f(v) = g(v) + h(v), \tag{8}$$

where $g(v)$ is the path cost from root node to node v, and $h(v)$ is an estimation of the minimum cost path from node v to a leaf node. If node v corresponds to

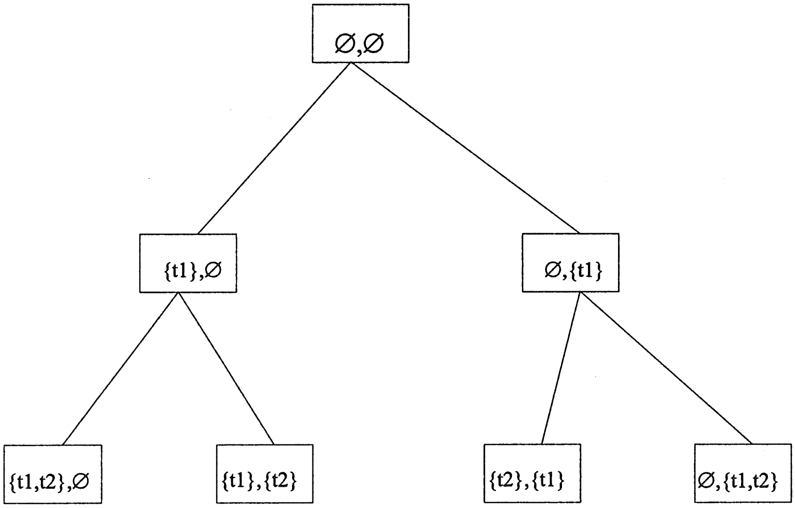

Figure 1. A state-space search tree for two modules to be assigned to two processors

an incomplete assignment with $t_1, t_2, \ldots, t_s$ assigned, then $g(v)$ is defined as

$$\begin{aligned} g(v) &= COST(X_v^s) \\ &= \sum_{k=1}^{n}\sum_{i=1}^{s} \lambda_k x_{ik} e_{ik} + \sum_{k=1}^{n-1}\sum_{b>k} - \sum_{i=1}^{s}\sum_{j=1}^{s} \mu_{kb} x_{ik} x_{jb} (c_{ij}/W_{kb}), \end{aligned} \tag{9}$$

which is a form of (6) when X is substituted by X_v^s. This gives the path cost from the root node to node v. We define $h(v)$, for a node v corresponding to a task assignment $t_1, t_2, \ldots, t_s$ as

$$h(v) = \Big[\sum_{i=s+1}^{m} \min_{k \in P} \Big(\lambda_k e_{ik} + \sum_{b \neq k} \sum_{j=1}^{s} \mu_{kb} x_{jb} (c_{ij}/W_{kb}) \Big) \Big]. \tag{10}$$

In (10), the first term reflects the unreliability caused by executing module t_i at processor p_k and the second term reflects the unreliability caused by the intermodule communication between t_i (as if t_i were assigned to p_k) and any of the already assigned modules.

To reduce the solution space, a node with incomplete solution is closed if it violates any of the system constraints or the cost of incomplete solution (i.e. $g(v)$) is greater than or equal to an upper bound (the upper bound is initially obtained by Algorithm 2 discussed later). The order in which modules are assigned (Step 2 of Algorithm 1) generally helps in early fathoming of live nodes which could not lead to a better solution (Mahmood, 1994). When a feasible solution with cost less than the upper bound is found, the upper bound is modified to the cost of that allocation and the allocation is saved as an *incumbent* allocation. The algorithm terminates when all the nodes are expanded.

The proposed algorithm generates optimal allocations for both constrained and unconstrained problems. This can be shown as follows. All the internal nodes in a state-space search tree that enumerates all possible allocations represent incomplete allocations. The proposed algorithm kills an internal node only if it violates any of the system constraints or value of the cost function for an incomplete allocation is less than the upper bound. Under no other condition, an internal node is excluded from further expansion/search. Since the cost function is a non- decreasing function as we move down the tree, any partial allocation that either violates a system constraint or value of the cost function for incomplete allocation is greater than the upper bound cannot lead to a better feasible solution. Therefore, the algorithm will find an optimal allocation.

Algorithm 1

```
1. Use Algorithm 2 to obtain an initial allocation $X$ and calculate $COST(X)$
   using (6). Save allocation $X$ as an incumbent solution and set upper-bound
   $= COST(X)$
2. Order modules in $T$ in non-decreasing values of $C_i$
3. Put root node $r = (\emptyset, \emptyset, \ldots, \emptyset)$ on a list called LIVE and set $f(r) = 0$.
4. While (LIVE list is not empty) do
     Find a node $v$ in the list LIVE with a minimum $f$ value.
     If $v$ is a leaf node and $COST(X) <$ upper-bound then
       Save state associated with node $v$ as incumbent solution and modify the
       upper bound to the cost of that assignment.
       Move node $v$ from list LIVE to another list called DEAD.
     Else
       Expand node $v$ by assigning the next module in the list to each of the
       $n$ processors to generate $n$ successor nodes of node $v$. Move all the invalid
       nodes to list DEAD.
       Move all nodes with $f$ value greater than upper-bound to DEAD and all
       the remaining nodes to LIVE.
   End (While)
5. Return the incumbent solution. {It is the optimal solution}
```

Note that the worst case time complexity of the algorithm is exponential. Also, the function $f(v)$ is quite complicated and involves a large number of computations. Thus, the algorithm is suitable only for relatively small systems. Therefore, there is a need for efficient heuristics to find suboptimal allocations in a reasonable computational time.

To devise heuristic algorithms, we first examine the cost function given by (6). The first term reflects the unreliability caused by the execution of modules on processors of varying reliability and the second term gives the unreliability caused by the intermodule communication. If the intermodule communication is relatively low, then reducing the unreliability caused by the intermodule communication becomes a major concern. On the other hand, if the system is homogenous, then first term becomes constant and the execution cost becomes dominant. However, in general, both execution and communication should be considered.

The next two algorithms are based on a simple heuristic that to reduce the system unreliability, the module with maximum intermodule communication cost and accumulative execution time should be assigned to the most reliable processor. To reduce the unreliability caused by intermodule communication, tightly coupled modules (modules heavily communicating with each other) should be assigned to the same processor or to neighboring processors with most reliable links.

To incorporate the above heuristic, Algorithm 2 assigns the modules one by one such that each time a module is assigned, the system unreliability is minimized. At Step 2, a module with maximum value of C_i is assigned to the most reliable processor. The modules that must also be assigned to the same processors due to a system constraint are so assigned (Step 3.1). Step 3.2 finds a module, which has not been assigned yet and is tightly coupled with the already assigned modules and assigns it to a processor such that the system unreliability is minimized. The complete algorithm is given below. The time complexity of the algorithm is $O(nm^2)$

Algorithm 2

```
1. Initialize $A = \emptyset$, $S_k = \emptyset$ (for $k = 1$ to $n$) {$A$ is a list of modules which have
   been assigned and $S_k$ is a list of modules assigned to processor $p_k$}
2. Find a task $t_i \in T$ with a maximum value of $C_i$ and assign it to processor $p_k$
   such that $e_{ik}\lambda_k$ is a minimum over all the processors and no system constraint
   is violated. Set $T = T - t_i$; $A = A + t_i$; $S_k = S_k + t_i$
3. While (NOT $T$ empty) do
   3.1. If there are some modules in $T$ that must also be assigned to processor $p_k$
        at which the last module is assigned, then assign all of them to $p_k$ and
        remove them from $T$, add them to $A$ and $S_k$
   3.2. Remove a task $t_i \in T$ such that value of (7) is maximum
   3.3. Assign $t_i$ to processor $p_k$ such that $e_{ik}\lambda_k + \sum_{all\, j \in A} \sum_{b \neq k} x_{jb} c_{ij} \mu_{kb} / W_{kb}$
        is a minimum over all the processors and allocation of $t_k$ to processor $p_k$
        does not violate any of the system constraints. Assign $t_i$ to processor $p_k$
        and set $T = T - t_i$; $A = A + t_i$; $S + k = S_k + t_i$
   End (While)
4. Return the allocation $S_k$, $k = 1$ to $n$.
```

Note that Step 3 of the above algorithm does not consider any intermodule communication cost. To improve on this, Algorithm 3 (below) generates n assignments by assigning the first module on each of the n different processors and then selects the best one from them as a solution. The time complexity of the algorithm is $O(n^2m^2)$

Algorithm 3

```
1. Initialize $min = A\_max\_number$, $A = \emptyset$, $S_k = \emptyset$ (for $k = 1$ to $n$) $\{A$ is a list
   of modules which have been assigned and $S_k$ is a list of modules assigned to
   processor $p_k\}$
2. For $k = 1$ to $n$ do
   2.1. Find a task $t_i \in T$ with a maximum value of $C_i$ and assign it to processor $p_k$.
        Set $T = T - t_i$; $A = A + t_i$; $S_k = S_k + t_i$
   2.2. Apply step 3 of Algorithm 2
   2.3. {Let $X_k$ be the allocation obtained by Step 2.1 and 2.2.}
        If $COST(X_k) <$ min then
          min = $COST(X_k)$; BEST = $X_k$
   End {for}
3. Return BEST
```

4. Simulation results

Heuristics do not always generate optimal solutions and it is difficult to study them analytically for the average and worst-case behavior (Wah, 1984). Therefore, evaluation of a heuristic is generally done by simulation and by comparing its solutions with the solutions obtained by other heuristics or with optimal solutions for small tractable problems (Ghosh, Murthy, and Moffett, 1992). Therefore, we carried out a simulation study to establish the validity and efficiency of our algorithms. We implemented our algorithms along with the Shatz's algorithm and performed a total of 294 simulation runs for both small systems containing 3–5 processors and large systems containing 12–20 processors. The number of modules was varied from 4 to 8 for small systems and from 10 to 35 for large systems.

We used test data similar to the one used by Shatz, Wang, and Goto (1992). The failure rate of a processor is in the range of 0.00005–0.00010 and the failure rate of communication link is in the range of 0.00015–0.00030. The AET of a module is in the range of 15–25 and the IMC is in the range of 0–C, where C varies from 15 to 24 for different systems. The system configurations were generated randomly and the only system constraint applied is the processing time limit, which was varied from 30 to 120 for different system configurations. We also generated about 10% of the total cases in which there were relatively small number of tightly coupled modules or the execution cost was dominant as compared to the communication cost.

For small systems, we performed 240 simulation runs for different system configurations to evaluate the efficiency of the heuristics. To measure the error of a heuristic algorithm A, we used the relative percentage error $r(A)$, given by

$$r(A) = \frac{R(opt) - R(A)}{R(opt)} \times 100, \tag{11}$$

where $R(opt)$ is the system reliability of the optimal allocation obtained by Algorithm 1, and $R(A)$ is the system reliability of the assignment generated by heuristic A.

Table 2 summarizes the performance of Algorithm 1. The data shown for each system configuration is an average over all the simulation runs for the particular configuration. The simulation results show that Algorithm 1 generated relatively small percentage of nodes to obtain optimal solutions. The simulation results showing the effectiveness of the heuristic algorithms are summarized in Table 3 and Fig. 2, using formats similar to those in Lo (1983). The reliability of assignments generated by Algorithm 3 has a relative error 1.8% as compared to 3.9% and 3.1% for Algorithm 2 and Shatz's algorithm respectively. Algorithm 3 found optimal allocations for 21.3% cases whereas Algorithm 2 and Shatz's algorithm found for 10.8% and 13.3% cases respectively. On average, Shatz's algorithm produced better allocation as compared to Algorithm 2. However, it was observed that Algorithm 2 outperformed Shatz's algorithm in those system configurations in which there were relatively fewer tightly coupled modules or execution cost was dominant as compared to communication cost. The results also show that Algorithm 2 generated assignments with relative error quite close

n	M	No. of nodes in complete search tree	No. of nodes generated by Algorithm 1	%
3	4	64	12	52.9%
3	6	1093	308	28.2%
3	8	9841	1227	12.5%
3	12	797161	1228	0.2%
4	6	5461	1333	24.4%
4	8	87381	9301	10.6%
5	6	19531	2511	12.9%
5	8	488281	20081	4.1%

Table 2. Number of nodes generated by Algorithm 1

Alg.	Avg r	$r = 0$	$r < 1$	$r < 2$	$r < 3$	$r < 4$	$r < 5$	$r < 6$	$r < 10$	worst r
2	3.9	10.8%	20.4%	32.9%	49.5%	63.3%	75.4%	87.1%	95.4%	14.5
3	1.8	21.3%	43.0%	66.3%	86.7%	93.0%	96.8%	98.9%	100%	7.3
Shatz	3.1	13.3%	31.6%	50.4%	67.1%	79.4%	84.4%	91.9%	96.1%	11.2

Table 3. Distribution of error r for different algorithms (small systems)

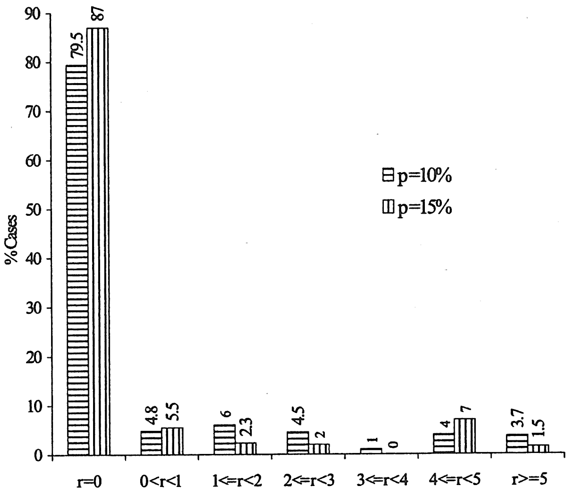

Figure 2. Number of cases for different values of p (Algorithm 2)

to the optimal values, therefore, its use in Algorithm 1 to obtain an upper-bound is a reasonable choice to reduce the solution search space.

For large systems (12–20 processors), the number of nodes expected to be generated by Algorithm 1 were very large. Therefore, we obtained solution for Algorithms 2, 3 and Shatz's and computed relative percentage error as

$$r(A) = \frac{R(A) - R(Shatz)}{R(A)} \times 100, \tag{12}$$

where $R(Shatz)$ is the system reliability of the allocation obtained by Shatz's algorithm and $R(A)$ is the system reliability of the assignment generated by algorithm A. Note that a negative value implies that the assignment generated by Shatz's algorithm is better than the one of algorithm A.

The results are summarized in Table 4 and Fig. 3. These results indicate the same tendency as the small system simulations. Algorithm 3 is the most effective of all the three heuristics we compared. There were only 9.3% cases for which Shatz's algorithm produced better results as compared to Algorithm 3. Shatz's algorithm outperformed Algorithm 2 for most of the cases. However, similar to small system simulation results, Algorithm 2 produced better assignments when there were relatively fewer strongly coupled modules, or execution

Alg.	$r < -2$	$r < -1$	$r < 0$	$r = 0$	$r < 1$	$r < 2$	$r < 3$	$r < 4$	$r < 5$	$r > 5$
2	18.5%	40.7%	64.8%	64.8%	81.5%	88.9%	88.9%	92.6%	94.5%	100%
3	3.7%	5.6%	9.3%	11.2%	26.0%	37.1%	51.9%	63.0%	83.4%	100%

Table 4. Distribution of error r for different algorithms (large systems)

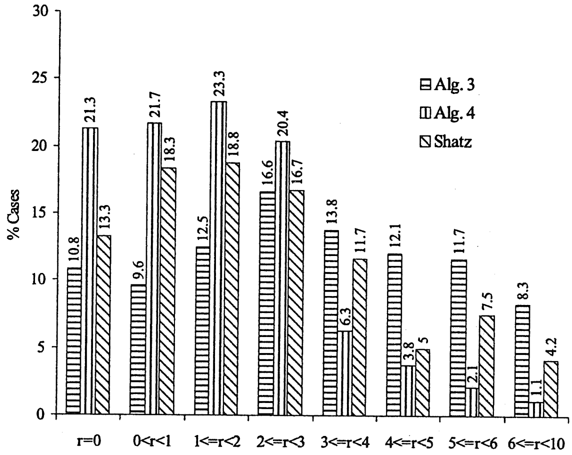

Figure 3. Number of cases for different ranges of r (small system)

cost was dominant as compared to communication cost. We also observed in our simulation study that the proposed algorithms generated the same module allocations for more than 70% of test runs. This percentage was even higher when failure rates of various links and/or processors were chosen to have different values. The explanation of this observation is that all the algorithms allocate modules based on the relative reliability of processors and links. If the failure rate of links have comparable values, the modules may be spread out to different processors; otherwise heavily communicating modules are allocated to the same processor. Similarly, more modules are allocated to those processors which are relatively more reliable.

5. Conclusions

In this paper, we considered the task allocation problem in distributed computing systems with the goal of maximizing system reliability. Here, first we presented a task allocation model similar to that of Shatz. The model is based on a cost function, which represents the unreliability caused by execution of modules on processors and the intermodule communication. Therefore, minimizing the cost function is equivalent to maximizing system reliability. We presented three algorithms: one to obtain optimal value and other two to obtain suboptimal value of cost function in a reasonable amount of computational time. Algorithm 1 can easily be tailored for network topologies other than considered in this paper by defining appropriate cost function and expressions for $g(v)$ and $h(v)$. Algorithm 2 and Algorithm 3 can also be used for other network topologies by defining an appropriate cost function and other related terms.

We demonstrated the efficiency and effectiveness of our algorithms through a simulation study in which we compared simulation results of our heuristics with those of the algorithm of Shatz. Algorithm 1 obtained optimal value by searching only a small percentage of the total search space. The results show that Algorithm 3 outperforms Shatz's algorithm in almost all the cases whereas Algorithm 2 outperforms Shatz's algorithm in almost all those cases in which there were relatively fewer strongly coupled modules or execution cost was relatively dominant as compared to the communication cost.

References

Bannister, J.A. and Trivedi, K.S. (1992) Task allocation in fault-tolerant distributed systems. *Acta Informatica*, **20**, 261–288.

Bokhari, S.H. (1988) Partitioning problems in parallel, pipelined and distributed computing. *IEEE Trans. Computers*, **37**, 48–57.

Bowen, N.S., Nikolous, C.N. and Ghafoor, A. (1992) On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems. *IEEE Trans. Computers*, **41**, 257–383.

Chu, W.W., Holloway, L.J. and Efe, K. (1980) Task allocation in distributed data processing. *IEEE Computer Mag.*, **13**, 57–691.

Coffman, E.G. (1976) *Computer and Job Scheduling Theory.* Wiley, New York.

Davoli, R., Giachini, L.A., Babaoglu, O., Amoroso, A. and Alvisi, L. (1996) Parallel computing in networks of workstations with Paralex. *IEEE Trans. Parallel and Dist. Systems*, **7**, 371–384.

Ghosh, D., Murthy, I. and Moffett, A. (1992) File allocation problem: comparison of models with worst and average communication delays. *Operations Research*, **40**, 1074–1085.

Hariri, S. and Raghavendra, C.S. (1986) Distributed functions allocation for reliability and delay optimization. *Proc. IEEE/ACM 1986 Fall Joint Computer Conference*, 344–352.

Kartik, S. and Murthy, C.R. (1997) Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Trans. Computers*, **46**, 719–724.

Kim, J., Lee, H. and Lee, S. (1997) Replicated process allocation for load distribution in fault-tolerant multicomputers. *IEEE Trans. Computers*, **46**, 499–505.

Lee, C.-H. and Shin, K.G. (1997) Optimal task assignment in homogeneous networks. *IEEE Trans. Parallel and Dist. Systems*, **8**, 119–129.

Lo, M.V. (1983) *Task assignment in distributed systems.* PhD Dissertation, Univ. of Illionois at Urbana-Champaign.

Ma, P.-Y.R. (1982) Task allocation model for distributed computing systems. *IEEE Trans. Computers*, **31**, 41–47.

Mahmood, A. (1994) *Adaptive File Allocation in Distributed Systems: A Cybernetics Approach.* PhD Thesis, University of London.

NILSSON, N.J. (1971) *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill.

SHATZ, S.M. and WANG, J.-P. (1987) An introduction to distributed software engineering. *IEEE Comp. Mag.*, **20**, 23–31.

SHATZ, S.M., WANG, J.P. and GOTO, M. (1992) Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Computers*, **41**, 1156–1168.

SHEN, C.C. and TSAI, W.H. (1985) A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Trans. Computers*, **34**, 197–203.

TAN, M. and SIEGAL, H.J. (1998) A stochastic model for heterogeneous computing and its application in data relocation scheme development. *IEEE Trans. on Parallel and Distributed Systems*, **9**, 1088–1101.

WAH, B.W. (1984) File placement on distributed computer systems. *IEEE Computer*, 23–32.