

# Biobjective Scheduling Algorithms for Execution Time–Reliability Trade-off in Heterogeneous Computing Systems\*

ATAKAN DOĞAN<sup>1</sup> AND FÜSUN ÖZGÜNER<sup>2</sup>

<sup>1</sup>*Department of Electrical and Electronics Engineering, Anadolu University, 26470 Eskişehir, Turkey*

<sup>2</sup>*Department of Electrical Engineering, The Ohio State University, 2015 Neil Avenue Columbus,  
OH 43210-1272, USA*

*Email: atdogan@anadolu.edu.tr, ozguner@ece.osu.edu*

A heterogeneous computing (HC) system is composed of a suite of geographically distributed high-performance machines interconnected by a high-speed network, thereby providing high-speed execution of computationally intensive applications with diverse demands. In HC systems, however, there is a possibility of machine and network failures and this can have an adverse impact on applications running on the system. In order to decrease the impact of failures on an application, matching and scheduling algorithms must be devised which minimize not only the execution time but also the failure probability of the application. However, because of the conflicting requirements, it is not possible to minimize both at the same time. Thus, the goal of this paper is to develop matching and scheduling algorithms which account for both the execution time and the failure probability and can trade off execution time against the failure probability of the application. In order to attain these goals, a biobjective scheduling problem is first formulated and then two different algorithms, the biobjective dynamic level scheduling algorithm and the biobjective genetic algorithm, are developed. Unique to both algorithms is the expression used for computing the failure probability of an application with precedence constraints. The simulation results confirm that the proposed algorithms can be used for producing task assignments where the execution time is weighed against the failure probability.

*Received 10 April 2004; revised 12 November 2004*

## 1. INTRODUCTION

Heterogeneous computing (HC) systems are among the emerging platforms for executing computationally intensive applications with diverse computing needs. An HC system is formed by interconnecting a collection of geographically distributed dissimilar machines by a high-speed network. However, there are still many challenges involved in building a functional HC system, e.g. achieving high-speed communication over a wide-area network, developing a middleware that enables a number of users to access all the available resources and services of the system in a transparent and efficient way, matching and scheduling problems [1, 2]. In particular, the matching and scheduling problem, which is the problem of minimizing the makespan of an application by scheduling its tasks to machines in the system, is an NP-hard problem and one of the challenges that has been studied

extensively in the literature. Many static, dynamic and even hybrid algorithms have been proposed to minimize the execution time of applications with precedence constraints running on an HC system, e.g. [3, 4, 5, 6].

Although advances in computer and networking technologies have made it possible to apply heterogeneous computing at a global scale, machine and network failures are inevitable in any large network of machines. An experiment conducted in [7] found that the estimated value of the mean time-to-failure of a machine connected to a local area network ranged from 4 to 33 days. In the literature several techniques have been developed to reduce the adverse effect of failures on applications executing on a distributed system. One approach is to employ a reliable scheduling algorithm in which the tasks of an application are assigned to machines in such a way that the failure probability of the application is minimized. The reliable scheduling technique has been pursued in allocating distributed programs, e.g. [8, 9, 10], undirected task graphs, e.g. [11, 12, 13] and directed acyclic task graphs, e.g. [14, 15]

\*A preliminary version of this paper was published in 2001 *International Parallel and Distributed Processing Symposium (IPDPS'01)*.

to distributed systems. In addition, reliability has been considered for real-time systems, e.g. [16, 17].

It was shown in [18] that a scheduling algorithm that minimizes only the schedule length may lead to a high failure probability, and that a reliable scheduling algorithm that minimizes only the failure probability may yield a high schedule length for an application running on an HC system. This result implies that a scheduling algorithm must account for both the execution time and the failure probability of an application. In addition, there are usually conflicting requirements between minimizing the execution time and the failure probability of an application, and it may not be possible to simultaneously minimize both. Consequently, a scheduling algorithm must be capable of balancing the execution time and failure probability of the application, i.e. it should be able to produce task assignments whereby the execution time is decreased at the expense of higher failure probability or vice versa. Unfortunately, there are few algorithms in the literature that address the problem of minimizing both the execution time and the failure probability of applications, or the problem of trading execution time for failure probability.

In [16] and similar studies which attempt to schedule application tasks with timing constraints, the primary goal is first to satisfy each task's timing constraint (deadline). Then, among the machines on which the task's deadline can be met, the task is scheduled to a machine where the failure probability of the application is minimized. Thus, in the framework of [16] and similar studies, the execution time of the application is not of any concern. In addition, even though there are two objectives, they are not considered simultaneously during the scheduling. Finally, there is no trading, i.e. missing the deadline of a task is not traded off against a lower failure probability of the application.

The study in [17] is unique in that scheduling decisions are based on maximizing an objective function that is a multiplication of two objective functions. These two objective functions are the probability that all tasks are completed before their respective deadlines and the reliability of the application. To optimize this composite objective function, an optimal algorithm is proposed in [17]. The main difference between the current study and [17] is the fact that minimizing execution time rather than meeting task deadlines is considered as one of the objectives here. Furthermore, the optimal algorithm of [17] seems to be too slow to be used for problems of practical size.

For applications with precedence constraints executing on an HC system, [19] is the first study to address the problem of minimizing both the execution time and the failure probability at the same time. In [19], the network topology of the HC system was assumed to be a tree. The method of [19] was extended to general network topologies in [20], which is the most relevant study to our own.

In summary, the current study is motivated by the fact that there is a trade-off between minimizing execution time and minimizing the failure probability of an application, and a scheduling algorithm needs to be developed to trade off between these two objectives. Since there are two

conflicting objectives, a biobjective scheduling problem is first formulated in Section 3, where the first objective is the schedule length and the second is the failure probability. In order to compute the schedule length and failure probability of an application with precedence constraints under a given task assignment, two mathematical models are derived in Sections 3.1 and 3.2. The model used to compute the failure probability is unique in the literature in that it is specifically formulated for HC systems with arbitrary network topology and that it is computationally efficient in estimating the failure probability. We should note here that the same two objectives are also considered in [20]. However, [20] does not provide a rigorous formulation of the biobjective scheduling problem as in Section 3. Furthermore, [20] proposes only a method to estimate the reliability of the communication between two machines, whereas the mathematical model presented in this study estimates the reliability of an application with precedence constraints. Mathematical models similar to the one developed here can be found in the literature, e.g. [14, 16, 17]. Studies [14], [16] and [17] actually use the model proposed in [11] for computing the reliability of an application. However, the model of [11] is specifically formulated for applications that can be modeled using undirected task graphs and is therefore not well suited to computing the failure probability of an application with precedence constraints. In addition, the model is applicable only to computing systems with tree network topology, although in [17] this limitation is overcome by arbitrarily choosing one of the paths among many possible ones between two tasks assigned to different machines to establish the intertask communication.

In order to solve the biobjective scheduling problem, two matching and scheduling algorithms, the biobjective dynamic level scheduling algorithm and biobjective genetic algorithm, are developed in Sections 4 and 5 respectively. The first algorithm is obtained by modifying an existing static matching and scheduling algorithm, and the second one is a standard genetic algorithm tuned to solve the biobjective scheduling problem. The extensive simulation studies presented in Section 6 show that both algorithms can be used to trade off execution time against the failure probability of applications.

## 2. MODELS AND ASSUMPTIONS

The network topology of an HC system is modeled using a connected, undirected graph  $G = (M, N)$ , where  $M$  denotes the computation and communication machines and  $N$  denotes the communication links. Let  $m_j \in M$  denote a machine, where  $1 \leq j \leq p + q$ , and  $n_{k,l} \in N$  denote a link between machines  $m_k$  and  $m_l$ . A machine ( $m_j$ ) will refer to a computation machine if  $1 \leq j \leq p$  and to a communication machine if  $p + 1 \leq j \leq p + q$ . Let  $\mathcal{R} = M \cup N$  denote the set of resources in an HC system; an element  $r_i \in \mathcal{R}$  refers to either a machine or a network link. The set  $\mathcal{R}$  is introduced only for notational convenience. A simple path  $p_{s,t}$  between machines  $m_s$  and  $m_t$  is defined as the set of resources that form a path from machine  $m_s$  to  $m_t$  in which

a resource is not visited more than once. The resource set includes both the source and destination machines as well. In this model of the system, computation machines are interconnected by a network of communication machines (e.g. switches, bridges, routers) and links. In addition, the network has an arbitrary topology and can accommodate different networking technologies.

An application executing on the system is represented using a directed acyclic graph (DAG)  $T = (V, E)$ , where  $V = \{v_1, v_2, \dots, v_n\}$  is the set of tasks to be executed and the set  $E$  of directed edges denotes the data communication between pairs of tasks. Let  $t_{i,j}^E$  be the expected execution time of task  $v_i$  on machine  $m_j$ ,  $1 \leq j \leq p$ . It is assumed that the expected execution time  $t_{i,j}^E$ ,  $1 \leq i \leq n$  and  $1 \leq j \leq p$ , is known. Techniques such as code profiling/analytic benchmarking [21] and statistical prediction [22] have been devised to estimate  $t_{i,j}^E$ . Let  $e_{k,l} \in E$  indicate communication from task  $v_k$  to task  $v_l$ , where task  $v_k$  ( $v_l$ ) is said to be an immediate predecessor (successor) task of task  $v_l$  ( $v_k$ ). Associated with directed edge  $e_{k,l} \in E$  is the volume of data in terms of bytes, which is denoted by  $d_{k,l}$ , that will be transmitted from task  $v_k$  to task  $v_l$  upon completion of task  $v_k$ .

Throughout the paper, it is assumed that all computation machines are dedicated, i.e. a task will run to completion without preemption on any computation machine in the system. Such a computing system is assumed to be controlled by a centralized scheduler that allocates system resources to applications with the goal of minimizing both the execution time and the failure probability of applications.

Regarding resource failures in the system, the following three assumptions are made. These assumptions are common to other studies, e.g. [11, 12, 13, 14], that deal with analyzing the reliability of computer systems. The failure of a resource in the system is assumed to follow a Poisson process and each resource  $r_i \in \mathcal{R}$  is accordingly associated with a constant failure rate  $\lambda_{r_i}$ . It should be noted that modeling the failure of a resource by a Poisson process may not always coincide with the actual failure dynamic of the resource. However, it is shown experimentally in [7] that such an assumption can still result in reasonably useful mathematical models. For mathematical tractability, failures of resources are assumed to be statistically independent. In addition, once a resource has failed, it is assumed that it remains in the failed state for the remainder of the execution of the application.

Finally, let  $\mathcal{M} : V \rightarrow M$  denote a matching function, where  $\mathcal{M}(i)$ ,  $1 \leq i \leq n$ , defines the machine to which task  $v_i$  is assigned. Note that only computation machines are considered for executing a given task. Thus,  $\mathcal{M}(i) = m_j$  implies that  $m_j$  is a computation machine ( $1 \leq j \leq p$ ). Let  $\mathcal{S}_j : V \rightarrow \{0, 1, \dots, n\}$  be a scheduling function, where  $\mathcal{S}_j(i)$ ,  $1 \leq i \leq n$  and  $1 \leq j \leq p$ , denotes the execution order of task  $v_i$  on machine  $m_j$  ( $\mathcal{S}_j(i) = 0$  indicates that task  $v_i$  is not assigned to machine  $m_j$ ). Consequently, a matching function  $\mathcal{M}$  and a set of scheduling functions  $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_p\}$  can represent a possible assignment of an application to machines in an HC system. Let  $\mathcal{X}_i = \{\mathcal{M}(i), \mathcal{S}_{\mathcal{M}(i)}(i)\}$  denote a possible matching and scheduling

decision for task  $v_i$  and  $\pi_i$  denote all possible matching and scheduling decisions for task  $v_i$ . Thus,  $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \times \mathcal{X}_n$  defines a possible task assignment of tasks in set  $V$  to machines in set  $M$  and  $\pi = \pi_1 \times \pi_2 \times \dots \times \pi_n$  represents all possible task assignments.

### 3. A BIOBJECTIVE SCHEDULING PROBLEM

As noted, in the context of this study, there are two objectives, namely minimizing execution time and minimizing failure probability, for which a Pareto-optimal [23] solution will be sought. Thus, a biobjective scheduling problem is defined as

$$\min_{\mathcal{X} \in \pi} \begin{bmatrix} J_1(\mathcal{X}) \\ J_2(\mathcal{X}) \end{bmatrix} \quad (1)$$

where  $J_1$  denotes the schedule length and  $J_2$  denotes the failure probability of an application under task assignment  $\mathcal{X}$ . It should be emphasized that the biobjective scheduling problem formulation of Equation (1) is new in the literature. In the following two sections, both objective functions  $J_1(\mathcal{X})$  and  $J_2(\mathcal{X})$  are formally defined.

#### 3.1. The first objective: minimize the schedule length

Suppose that task assignment  $\mathcal{X}$  is given. Because of the precedence constraints, task  $v_i$  cannot start running on machine  $\mathcal{M}(i)$  unless all data items from its immediate predecessor tasks have been received by machine  $\mathcal{M}(i)$ . Let  $t_{i,k}^D$  ( $e_{k,i} \in E$ ) denote the time when task  $v_i$  has received the data from task  $v_k$  and

$$t_{i,k}^D = \begin{cases} t_k^F, & \text{if } \mathcal{M}(k) = \mathcal{M}(i) \\ t_k^F + d_{k,i} c_{\mathcal{M}(k), \mathcal{M}(i)}, & \text{otherwise} \end{cases} \quad (2)$$

where  $t_k^F$  denotes the finish time of task  $v_k$  and  $c_{s,t}$  denotes the expected transmission time of sending one byte of data from machine  $m_s$  to machine  $m_t$ . The time when all data items of task  $v_i$  have been received by machine  $\mathcal{M}(i)$  is referred to as the data arrival time. Definition 1 formalizes the data arrival time of a task.

**DEFINITION 1.** *The data arrival time of task  $v_i$ , which is denoted by  $t_i^D$ , is defined to be*

$$t_i^D = \max_{e_{k,i} \in E} \{t_{i,k}^D\} \quad (3)$$

In order for the execution of a task to start on a machine, all data items for the task must have been received and the machine must be available. Thus, the start time of task  $v_i$ , which is denoted by  $t_i^S$ , is defined to be

$$t_i^S = \max \{t_i^M, t_i^D\} \quad (4)$$

where  $t_i^M$  denotes the time when machine  $\mathcal{M}(i)$  will be available to execute task  $v_i$  ( $t_i^M = 0$  if task  $v_i$  is the first task to be executed on machine  $\mathcal{M}(i)$ ) and  $t_i^M$  equals the finish time of the  $(k-1)$ th task if it is the  $k$ th task. Finally, the finish time of task  $v_i$  ( $t_i^F$ ) is defined to be

$$t_i^F = t_i^S + t_{i, \mathcal{M}(i)}^E \quad (5)$$

Thus, the schedule length of the application under task assignment  $\mathcal{X}$  is given by

$$J_1(\mathcal{X}) = \max_{v_i \in V} \{t_i^F\} \quad (6)$$

### 3.2. The second objective: minimize the failure probability

In this section, a new mathematical model to compute the reliability of an application with precedence constraints is presented. Before the presentation of this model some notation needs to be introduced. Let  $R_j(T, \mathcal{X})$ ,  $1 \leq j \leq p$ , denote the reliability of computation machine  $m_j$ , which is the probability that machine  $m_j$  is functional for the execution of tasks assigned to it under task assignment  $\mathcal{X}$ . In addition, let  $R_j(T, \mathcal{X})$ ,  $p + 1 \leq j \leq p + q$ , denote the reliability of communication machine  $m_j$  and  $R_{k,l}(T, \mathcal{X})$  denote the reliability of link  $n_{k,l}$ , where the reliability of a communication resource (machine or link) is the probability that the communication resource is functional for performing intertask communication during the execution of the application under task assignment  $\mathcal{X}$ . Note that, since the failure of a resource is governed by a Poisson process, the reliability of resource  $r_i$  at time  $t$  is  $e^{-\lambda_i t}$  [24].

Successful completion of the execution of the application requires that each computation machine be functional during the time that its assigned tasks are executing and that each communication resource that will be used in intertask communication be functional during the time that the intertask communication is taking place; it thus depends on the reliability of the resources to which the application is allocated. The probability that application  $T$  can run successfully on an HC system under task assignment  $\mathcal{X}$  is denoted by  $R(T, \mathcal{X})$ , which also represents the reliability of the HC system when application  $T$  is allocated by  $\mathcal{X}$ . Assuming that the failures of resources are statistically independent,  $R(T, \mathcal{X})$  is defined to be

$$\begin{aligned} R(T, \mathcal{X}) &= \prod_{m_j \in \mathcal{R}_K} R_j(T, \mathcal{X}) \cdot \prod_{n_{k,l} \in \mathcal{R}_K} R_{k,l}(T, \mathcal{X}) \\ &= \prod_{m_j \in \mathcal{R}_K} e^{-\lambda_j t_j^A} \cdot \prod_{n_{k,l} \in \mathcal{R}_K} e^{-\lambda_{k,l} t_{k,l}^A} \\ &= e^{(-\text{COST}(\mathcal{X}))} \end{aligned} \quad (7)$$

where  $\text{COST}(\mathcal{X}) = \sum_{m_j \in \mathcal{R}_K} \lambda_j t_j^A + \sum_{n_{k,l} \in \mathcal{R}_K} \lambda_{k,l} t_{k,l}^A$ , and  $\lambda_j$  and  $\lambda_{k,l}$  are the failure rates of machine  $m_j$  and link  $n_{k,l}$  respectively. In addition,  $t_j^A$ ,  $1 \leq j \leq p$ , denotes the time at which computation machine  $m_j$  will complete the execution of tasks under  $\mathcal{X}$  and is defined to be

$$t_j^A = \max_{S_j(i) > 0} \{t_i^F\} \quad (8)$$

Variables  $t_j^A$ ,  $p + 1 \leq j \leq p + q$ , and  $t_{k,l}^A$  denote the times at which communication machine  $m_j$  and link  $n_{k,l}$  will complete the intertask data communication under  $\mathcal{X}$

respectively, and are defined to be

$$\begin{aligned} t_j^A &= \max_{e_{u,v} \in E} \left\{ \mathcal{I}_j(\mathcal{M}(u), \mathcal{M}(v)) t_{v,u}^D \right\} \quad \text{and} \\ t_{k,l}^A &= \max_{e_{u,v} \in E} \left\{ \mathcal{I}_{k,l}(\mathcal{M}(u), \mathcal{M}(v)) t_{v,u}^D \right\} \end{aligned} \quad (9)$$

where

$$\begin{aligned} \mathcal{I}_j(s, t) &= \begin{cases} 1, & m_j \text{ is on } p_{s,t} \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \\ \mathcal{I}_{k,l}(s, t) &= \begin{cases} 1, & n_{k,l} \text{ is on } p_{s,t} \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Finally,  $\mathcal{R}_K$  is a resource set that is composed of computation and communication machines and links. That is,  $\mathcal{R}_K$  is a subset of resources used for executing the application under  $\mathcal{X}$  and determined with respect to set  $K = \{m_j \mid m_j = \mathcal{M}(i) \text{ and } v_i \in V\}$ , which is the set of computation machines to which at least one task of the application is allocated. If the network topology of an HC system can be modeled by a tree (i.e. there exists a unique simple path between any two machines) set  $\mathcal{R}_K$  is determined as follows. (i) Include all computation machines to which at least one task is assigned into  $\mathcal{R}_K$ . (ii) For all  $e_{k,l} \in E$ , include all communication machines and links that form the unique simple path from machine  $m_s$  to  $m_t$  into  $\mathcal{R}_K$  provided tasks  $v_k$  and  $v_l$  are executing on machines  $m_s$  and  $m_t$  respectively. If the network topology is not a tree, computing  $\mathcal{R}_K$  is non-trivial; this is addressed in Section 3.2.1.

As a result, the failure probability of the application under task assignment  $\mathcal{X}$  is defined as

$$J_2(\mathcal{X}) = 1 - R(T, \mathcal{X}) \quad (10)$$

#### 3.2.1. K-terminal reliability computation

In the previous section, it was noted that the computation of set  $\mathcal{R}_K$  is a difficult task for a system with arbitrary network topology. Basically, the complexity of this task comes from the fact that if there exists more than one simple path between two machines in the system, it is not clear which resources must be included into  $\mathcal{R}_K$  to represent the communication between the machines. In this section, a computationally simple method is presented to compute set  $\mathcal{R}_K$ . This method is based on the computation of the K-terminal reliability, which is formally defined below.

**DEFINITION 2.** *The K-terminal reliability is the probability that all machine pairs in a set  $K$ , which is a subset of computation machines in the system, can communicate.*

For the computation of the K-terminal reliability, K-trees are used [25]. Let  $K = \{m_1, m_2, \dots, m_k\}$ ,  $2 \leq k \leq p$ , be any subset of  $M$ , where  $m_j \in K$  is a computation machine. A K-tree is defined as follows.

**DEFINITION 3.** *A K-tree of  $G = (M, N)$  with respect to a set of computation machines  $K$  is a tree in which leaf vertices of the tree correspond to computation machines in set  $K$  and the others correspond to only communication machines.*

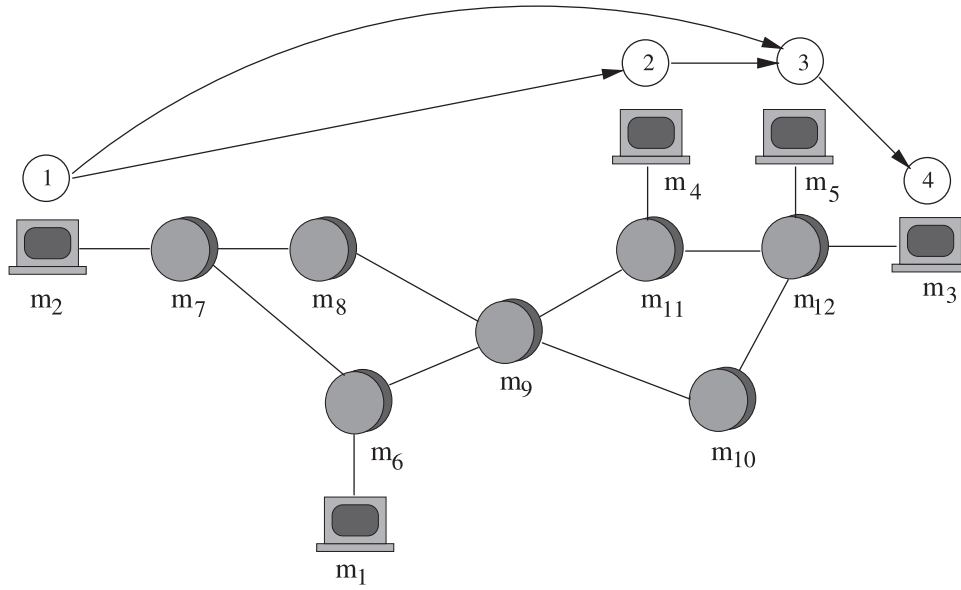


FIGURE 1. An example allocation of a task graph on an HC system.

Based on K-trees, the K-terminal reliability is the probability that there exists at least one functional K-tree in the system. In the following, the K-terminal reliability is formally derived.

Let  $T^i$  denote the  $i$ th K-tree of  $G$ , with respect to  $K$ , and  $T^i$  be represented as a set of resources  $T^i = \{r_1^i, r_2^i, \dots, r_{l_i}^i\}$ , where  $l_i$  is the number of resources in the  $i$ th K-tree. Let  $\mathcal{E}$  be the event that at least one K-tree is failure-free and  $E_i$  be the event that the  $i$ th K-tree is failure-free. Using the inclusion-exclusion principle, the K-terminal reliability can be computed as

$$\begin{aligned}
 P[\mathcal{E}] &= P[E_1 \cup \dots \cup E_\gamma] \\
 &= \sum_{i=1}^{\gamma} P[E_i] - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} P[E_i \cap E_j] \\
 &\quad + \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} P[E_i \cap E_j \cap E_k] - \dots \\
 &\quad + (-1)^{\gamma-1} P[E_1 \cap \dots \cap E_\gamma] \quad (11)
 \end{aligned}$$

where  $\gamma$  is the number of different K-trees. In [18], we show that

$$\begin{aligned}
 P[\mathcal{E}] &= \sum_{i=1}^{\gamma} e^{-(\sum \Lambda^i)t} - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} e^{-(\sum (\Lambda^i \cup \Lambda^j))t} \\
 &\quad + \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} e^{-(\sum (\Lambda^i \cup \Lambda^j \cup \Lambda^k))t} - \dots \\
 &\quad + (-1)^{\gamma-1} e^{-(\sum (\Lambda^1 \cup \dots \cup \Lambda^\gamma))t} \quad (12)
 \end{aligned}$$

where  $\Lambda^i = \{\lambda_{r_1}^i, \dots, \lambda_{r_{l_i}^i}\}$  denotes the set of failure rates of the resources which form the  $i$ th K-tree and  $\sum \{\Lambda\}$  denotes the summation of the elements of  $\Lambda$ . However, computing

the exact K-terminal reliability for an arbitrary network using Equation (12) is NP-hard [26]. To simplify the computation of  $P[\mathcal{E}]$ ,  $\bar{e}^x$  can be replaced by its small-value approximation  $1 - x$ . After the substitution, it is shown in [18] that the K-terminal reliability expression (12) simplifies to

$$\begin{aligned}
 P[\mathcal{E}] &= 1 - \left[ \sum \left\{ \prod_{i=1}^{\gamma} \Lambda^i \right\} \right] t \\
 &= 1 - \sum_{r_i \in \mathcal{R}_K} \lambda_{r_i} t \quad (13)
 \end{aligned}$$

where  $\mathcal{R}_K = \{r_x \mid r_x \in \bigcap_{i=1}^{\gamma} T^i\}$ , i.e.  $\mathcal{R}_K$  is a set of resources that are common to all K-trees. For example, suppose that a DAG with four tasks will run on an HC system with five machines, as shown in Figure 1, where task  $v_1$  is assigned to machine  $m_2$ , and so on. With respect to the task assignment in Figure 1,  $K = \{m_2, m_3, m_4, m_5\}$  and there are only four distinct K-trees that span computation machines in set  $K$ . These are shown in Figure 2. Given the K-trees in Figure 2,  $\mathcal{R}_K = \{m_2, m_3, m_4, m_5, m_7, m_9, m_{11}, m_{12}, n_{2,7}, n_{3,12}, n_{4,11}, n_{5,12}\}$ . Note that the error due to the small-value approximation is less than  $\sum_{r_i \in \mathcal{R}_K} \lambda_{r_i} t$ .

Enumeration of all K-trees of an arbitrary network graph to compute set  $\mathcal{R}_K$  is computationally expensive even though it is possible to generate a K-tree of an undirected graph in  $O(|M||N|)$  [25]. For the efficient computation of  $\mathcal{R}_K$ , a new method which avoids the enumeration of K-trees is developed in this paper. The new method is based on Theorem 1, whose proof can be found in [18]. Before the presentation of this theorem, let  $p_{s,t}^i = \{r_1^i, r_2^i, \dots, r_{h_i}^i\}$  denote the  $i$ -th simple path from machine  $m_s$  to machine  $m_t$ , where  $h_i$  is the number of resources in the  $i$ -th simple path, and let  $\gamma_{s,t}$  denote the number of different simple paths between machines  $m_s$  and  $m_t$ .

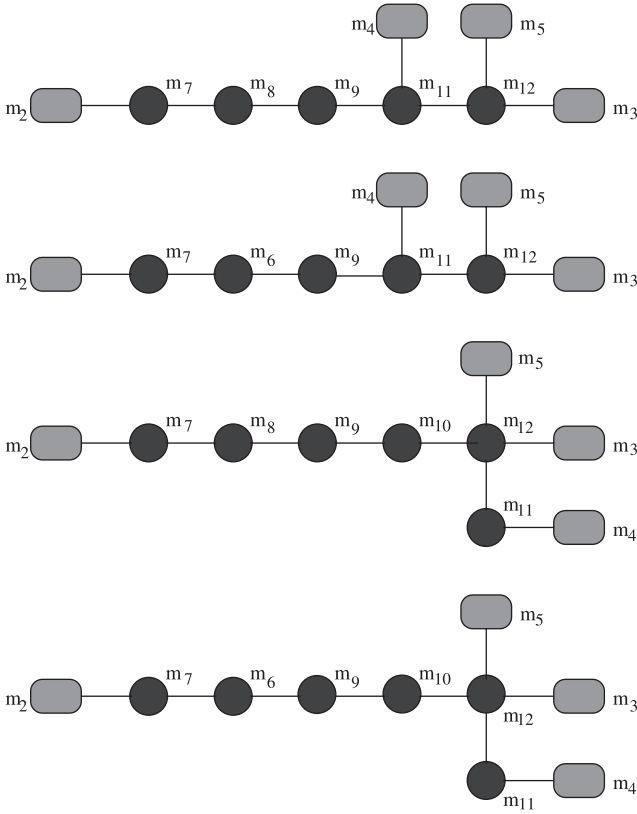


FIGURE 2. K-trees of  $K = \{m_2, m_3, m_4, m_5\}$ .

**THEOREM 1.** *The reliability of the communication between any machine pair  $(m_s, m_t)$  in the system, which is denoted by  $R_{\tau_{s,t}}(t)$ , can be approximated as*

$$R_{\tau_{s,t}}(t) = 1 - \sum_{r_i \in \tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}} \lambda_{r_i} t \quad (14)$$

where  $\tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}} = \{r_x | r_x \in \bigcap_{i=1}^{\gamma_{s,t}} p_{s,t}^i\}$ , i.e.  $\tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}$  is composed of resources that are common to all simple paths between machines  $m_s$  and  $m_t$ . In addition,  $\tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}$  can be identified in  $O(|M| + |N|)$  time.

For example, in order to compute the reliability of the communication between machines  $m_2$  and  $m_4$  in Figure 1, only the reliability of resources in set  $\tilde{\mathcal{R}}_{2,4}^4 = \{m_2, n_{2,7}, m_7, m_9, m_{11}, n_{11,4}, m_4\}$  needs to be considered. Note that each simple path between machines  $m_2$  and  $m_4$  includes the resources in  $\tilde{\mathcal{R}}_{2,4}^4$ , where  $p_{2,4}^1 = \{m_2, n_{2,7}, m_7, n_{7,8}, m_8, n_{8,9}, m_9, n_{9,11}, m_{11}, n_{11,4}, m_4\}$ ,  $p_{2,4}^2 = \{m_2, n_{2,7}, m_7, n_{7,6}, m_6, n_{6,9}, m_9, n_{9,11}, m_{11}, n_{11,4}, m_4\}$ ,  $p_{2,4}^3 = \{m_2, n_{2,7}, m_7, n_{7,8}, m_8, n_{8,9}, m_9, n_{9,10}, m_{10}, n_{10,12}, m_{12}, n_{12,11}, m_{11}, n_{11,4}, m_4\}$  and  $p_{2,4}^4 = \{m_2, n_{2,7}, m_7, n_{7,6}, m_6, n_{6,9}, m_9, n_{9,10}, m_{10}, n_{10,12}, m_{12}, n_{12,11}, m_{11}, n_{11,4}, m_4\}$ . The following theorem, whose proof can be found in [27], shows how to determine set  $\mathcal{R}_K$  without enumerating all K-trees.

**THEOREM 2.** *If  $\mathcal{R}_K = \{r_x | r_x \in \bigcap_{i=1}^{\gamma} \mathcal{T}^i\}$  and  $\tilde{\mathcal{R}}_K = \{r_x | r_x \in \bigcup_{i=2}^{|K|} \bigcup_{j=1}^{i-1} \tilde{\mathcal{R}}_{i,j}^{\gamma_{i,j}}\}$ , then  $\mathcal{R}_K \equiv \tilde{\mathcal{R}}_K$ . In addition, the time complexity of computing  $\mathcal{R}_K$  is  $O((|M| + |N|)|K|^2)$ .*

With respect to Theorem 2,  $\mathcal{R}_K$  can be formed by including the resources in set  $\tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}$  for each possible machine pair  $(m_s, m_t)$  in set  $K$  into  $\mathcal{R}_K$ . As a result, it is possible for the resources in  $\tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}$  to be included into  $\mathcal{R}_K$  even though there is no communication from machine  $m_s$  to  $m_t$  during the execution of the application. Thus, one may argue that some resources that will not be used for the execution of the application can appear in  $\mathcal{R}_K$ . However, according to Theorem 3, whose proof can be found in [27], all resources in  $\mathcal{R}_K$  will take part in the execution of the application.

**THEOREM 3.** *If  $\tilde{\mathcal{R}}_K = \bigcup_{e_{k,l} \in E} \tilde{\mathcal{I}}_{k,l}(s, t) \tilde{\mathcal{R}}_{s,t}^{\gamma_{s,t}}$ , where*

$$\tilde{\mathcal{I}}_{k,l}(s, t) = \begin{cases} 1, & m_s = \mathcal{M}(k) \text{ and } m_t = \mathcal{M}(l) \\ 0, & \text{otherwise.} \end{cases}$$

then  $\mathcal{R}_K \equiv \tilde{\mathcal{R}}_K$ . In addition, the time complexity of computing  $\mathcal{R}_K$  is  $O((|M| + |N|)|E|)$ .

#### 4. BIOBJECTIVE DYNAMIC LEVEL SCHEDULING ALGORITHM

The dynamic level scheduling (DLS) algorithm is a static list scheduling heuristic that has been developed to allocate a DAG-structured application to a set of heterogeneous machines to minimize the execution time of the application [3]. At each scheduling step, the DLS algorithm chooses the next task to schedule and the machine on which that task is to be executed by finding the ready task and machine pair that has the highest dynamic level. The dynamic level of a task-machine pair, which is denoted by  $DL(v_i, m_j)$ , is defined to be

$$DL(v_i, m_j) = SL(v_i) - \max\{t_i^M, t_i^D\} + \Delta(v_i, m_j) \quad (15)$$

The first term in Equation (15) is called the static level of task  $v_i$  and is defined to be

$$SL(v_i) = \hat{t}_i^E + \max_{e_{i,l} \in E} \{SL(v_l)\} \quad (16)$$

where  $\hat{t}_i^E$  denotes the median execution time of task  $v_i$  across all machines. The static level indicates the importance of a task in the precedence hierarchy by giving higher priority to tasks for which the time spent to complete the execution of the application is expected to be larger. The max term in Equation (15) defines the time when task  $v_i$  can begin execution on machine  $m_j$ . A task-machine pair with an earlier starting time will have higher scheduling priority. The third term in Equation (15) accounts for the machine speed differences and is defined to be

$$\Delta(v_i, m_j) = \hat{t}_i^E - t_{i,j}^E \quad (17)$$

If machine  $m_j$  runs task  $v_i$  faster than the other machines in the network,  $\Delta(v_i, m_j)$  will be positive, which increases the scheduling priority.

While making matching and scheduling decisions, the DLS algorithm does not account for the reliability of the resources in a HC system. In this study, the

1. Find all ready tasks and put them on *Ready* list;
2. **while** (all tasks in *Ready* list are scheduled)
3. **for** each  $v_i \in \text{Ready}$
4.   **for** each machine  $m_j \in M, j \leq p$
5.     Compute  $DL(v_i, m_j)$  and put on *List*<sub>1</sub>;
6.     Compute  $\Delta\text{COST}(v_i, m_j)$  and put on *List*<sub>2</sub>;
7.   **endfor**.
8. **endfor**.
9. Sort *List*<sub>1</sub> in decreasing order of  $DL(v_i, m_j)$ ;
10. Sort *List*<sub>2</sub> in increasing order of  $\Delta\text{COST}(v_i, m_j)$ ;
11. Find the task-machine pair  $(v_i^*, m_j^*)$  which minimizes  $\text{Rank}_{i,j}$ ;
12. Assign task  $v_i^*$  on machine  $m_j^*$  and then, update *Ready* list;
13. **endwhile**.

**FIGURE 3.** The biobjective dynamic level scheduling algorithm.

DLS algorithm is modified as shown in Figure 3 and the new algorithm will be referred to as the biobjective dynamic level scheduling (BDLS) algorithm. In the BDLS algorithm, when the dynamic level of a task-machine pair  $DL(v_i, m_j)$  is computed, the corresponding incremental cost  $\Delta\text{COST}(v_i, m_j)$  is also computed. The incremental cost of executing task  $v_i$  on machine  $m_j$  is defined to be

$$\begin{aligned} \Delta\text{COST}(v_i, m_j) &= \text{COST}(\tilde{\mathcal{X}}_i) - \text{COST}(\tilde{\mathcal{X}}_{i-1}) \\ &= \ln(R(T, \tilde{\mathcal{X}}_{i-1})) - \ln(R(T, \tilde{\mathcal{X}}_i)) \quad (18) \end{aligned}$$

where  $\tilde{\mathcal{X}}_i$  denotes a partial task assignment in which tasks  $\{v_1, v_2, \dots, v_i\}$  are assigned. Note that since  $R(T, \tilde{\mathcal{X}}_{i-1}) > R(T, \tilde{\mathcal{X}}_i)$  always holds,  $\Delta\text{COST}(v_i, m_j) > 0$ , i.e. making a new scheduling decision will always increase the failure probability of the application. Thus, the incremental cost must be kept low for each task in order to decrease the failure probability of the application.

According to Figure 3,  $DL(v_i, m_j)$  and  $\Delta\text{COST}(v_i, m_j)$  are computed for each task-machine pair  $(v_i, m_j)$ , where  $v_i$  is a ready task, and the results are stored in lists *List*<sub>1</sub> and *List*<sub>2</sub> respectively. Then, list *List*<sub>1</sub> is sorted in decreasing order and list *List*<sub>2</sub> is sorted in increasing order. Thus two ranks,  $\text{Rank}_{i,j}^1$  and  $\text{Rank}_{i,j}^2$ , are associated with each task-machine pair  $(v_i, m_j)$ , where  $\text{Rank}_{i,j}^1$  and  $\text{Rank}_{i,j}^2$  denote the orders of task-machine pair  $(v_i, m_j)$  in *List*<sub>1</sub> and *List*<sub>2</sub> respectively. Thus, a higher ranked task-machine pair in either list implies a better choice with respect to the corresponding objective. In the DLS algorithm, the task-machine pair  $(v_i^*, m_j^*)$  for which  $\text{Rank}_{i,j}^1 = 1$  is found, and task  $v_i^*$  is assigned to machine  $m_j^*$ . In the BDLS algorithm, however, the task-machine pair  $(v_i^*, m_j^*)$  for which  $\text{Rank}_{i,j}$  is minimized is taken as the current best scheduling decision.  $\text{Rank}_{i,j}$  is defined to be

$$\text{Rank}_{i,j} = \delta^1 \text{Rank}_{i,j}^1 + \delta^2 \text{Rank}_{i,j}^2 \quad (19)$$

where  $\delta^1$  and  $\delta^2$  are the weights that are introduced for trading off execution time for failure probability. Note that setting  $\delta^2 = 0$  reduces BDLS to DLS in that the goal of minimizing the failure probability is neglected, and setting  $\delta^1 = 0$  makes BDLS ignore the goal of minimizing the schedule length.

At this point, it is important to make a comparison between the BDLS algorithm and the RDLS algorithm proposed in [20], which is the only other algorithm in the literature that addresses the biobjective scheduling problem posed in this study. First, both the BDLS and RDLS algorithms are based on the DLS algorithm. Thus, the dynamic level expression of Equation (15) is common to both.

In order to assign a rank to a scheduling decision represented by a task-machine pair  $(v_i, m_j)$ , the BDLS creates two lists as explained above. On the other hand, the RDLS creates only one list based on the following expression:  $DL(v_i, m_j) - C(v_i, m_j)$ , where  $DL(v_i, m_j)$  is from Equation (15) and  $C(v_i, m_j)$  is a term that reflects the impact of scheduling task  $v_i$  on machine  $m_j$  on the failure probability of the application. Note that the higher the value of  $C(v_i, m_j)$ , the higher the failure probability. An important disadvantage of the rank assignment mechanism of the RDLS is the fact that  $DL(v_i, m_j)$  can dominate  $C(v_i, m_j)$  or vice versa. This is because the ranges of values that  $DL(v_i, m_j)$  and  $C(v_i, m_j)$  can take are different. As a result, during the scheduling, one of the objectives can be easily overlooked by the RDLS. In order to alleviate this problem in the RDLS, both objective functions must be normalized and then combined. Note that the BDLS does not have this problem simply because a rank from the corresponding list is associated with each objective and then they are combined. On the other hand, the BDLS allows one to make one objective dominate the other simply by setting the weights appropriately.

Another important distinction of the BDLS comes from the definition of  $\Delta\text{COST}(v_i, m_j)$  in Equation (18) compared with  $C(v_i, m_j)$  of the RDLS.  $\Delta\text{COST}(v_i, m_j)$  is an estimate of the difference in the reliability of the application due to the scheduling decision  $(v_i, m_j)$ . In order to compute  $\Delta\text{COST}(v_i, m_j)$ , a unique method based on the K-terminal reliability estimation was developed in the previous section.  $C(v_i, m_j)$ , on the other hand, is expressed in time units as a function of the reliability of communication between a source machine where the owner of the application is located and machine  $m_j$ . Thus, it is clear that  $\Delta\text{COST}(v_i, m_j)$  provides much better information on the impact of a scheduling decision on the failure probability of the application than  $C(v_i, m_j)$ .

It should be noted that it was in the framework of this study that the idea of trading off execution time against failure probability was first brought up. In addition, it is also possible to modify the objective function of the RDLS, i.e.  $DL(v_i, m_j) - C(v_i, m_j)$ , to enable such trading. Apparently, however, the concern in [20] was to produce compromise solutions by perturbing  $DL(v_i, m_j)$  via  $C(v_i, m_j)$  rather than trading execution time for failure probability.

## 5. A BIOBJECTIVE GENETIC ALGORITHM

Many approaches to genetic algorithms (GAs) have been proposed in the literature. In this study, however, a standard GA [28] is implemented: (i) generate an initial population; (ii) evaluate the fitness of each chromosome; (iii.a) repeat

(1) selection, (2) crossover, (3) mutation, (4) evaluate the fitness of each chromosome, (iii.b) until the population has converged. Details of the steps for the implementation of the proposed biobjective genetic algorithm (BGA) will be discussed in the following sections.

**5.1. Chromosome structure**

A chromosome is treated as a data structure into which a solution of the biobjective scheduling problem will be encoded. In the BGA, a chromosome  $\mathcal{C}_i$  is composed of  $p$  scheduling lists, where  $\mathcal{C}_i$  denotes the  $i$ -th chromosome. A scheduling list  $L_i^j$  includes all tasks assigned to a particular machine and defines the order of execution of the tasks on the machine, where  $L_i^j$  is the  $j$ -th scheduling list of the  $i$ -th chromosome.

A set of chromosomes is referred to as a population. The population size denoted by  $N_p$  is defined as the number of chromosomes in a population. In the BGA, the population size is kept fixed at  $N_p$  through the generations of the population.

In a population of the BGA, there can be two kinds of chromosome, namely valid chromosomes, each of which represents a possible execution of the application, and invalid chromosomes, each of which corresponds to a schedule under which the execution of the application is impossible. As also pointed out in below, an invalid chromosome can be due to the initial population generation phase, the crossover phase or the mutation phase. Note that ascertaining the validity of a chromosome takes  $O(|V| + |E|)$  time by using the depth-first search algorithm [29]. In a population of  $N_p$  chromosomes, a small number of invalid chromosomes are allowed in the population, where the maximum number of invalid chromosomes allowed is denoted by  $N_{iv}$ . The invalid chromosomes are differently treated from the valid ones only in the selection phase, which is explained in Section 5.4.

**5.2. Initial population generation**

In the BGA, an initial population from which the search for a Pareto-optimal solution starts is randomly generated as follows. (i) The first chromosome is generated: (a) the DAG is topologically sorted [29]; (b) each task is assigned to a randomly chosen machine in the order dictated by the topological sort. (ii) A new chromosome other than the first is generated: one of the previously generated chromosomes is randomly picked and mutated a random number of times (between one and the number of tasks) using the mutation operator, which will be defined below. (iii) If the newly generated chromosome is identical to any of the previously generated ones, it is discarded. The second and third steps are repeated until  $N_p$  unique chromosomes are generated. Note that there is no guarantee that a newly generated chromosome will be a valid one. The initial population generation algorithm is inspired by [30].

**5.3. Fitness of a chromosome**

In the area of evolutionary-based multiobjective optimization, determining the fitness values of chromosomes is one

of the major research problems [23]. Thus, in the literature, a number of fitness assignment techniques have been proposed, one of which is the sum of weighted global ratios (SWGR) [31]. Because of its high performance and easy implementation, proven in [31], the SWGR is implemented as the fitness assignment technique in the BGA.

According to the SWGR, the fitness value of a valid chromosome  $\mathcal{C}_i$ , which is denoted by  $fitness_i$ , is determined by aggregating two objective functions as follows:

$$fitness_i = \sum_{j=1}^2 \delta^j \frac{fit\_val_i^j - min\_fit^j}{max\_fit^j - min\_fit^j} \tag{20}$$

where  $fit\_val_i^j$  is the fitness of  $\mathcal{C}_i$  with respect to the  $j$ th objective,  $max\_fit^j$  and  $min\_fit^j$  are the best and worst fitness values encountered for the  $j$ th objective respectively and  $\delta^j$  is the weight for the  $j$ th objective as defined before. Note that  $fit\_val_i^1 = J_1(\mathcal{X})$  and  $fit\_val_i^2 = J_2(\mathcal{X})$ , where  $\mathcal{X}$  is encoded in chromosome  $\mathcal{C}_i$ . In addition, the greater  $fitness_i$ , the better the solution. Thus, if  $\mathcal{C}_i$  is an invalid chromosome,  $fitness_i$  is set to  $-\infty$ .

The BGA also implements elitism after the fitness evaluation process. Elitism is important because it ensures that the quality of the best solution found over generations is monotonically increasing. The details about incorporating elitism into a GA can be found in [18, 30, 32] and thus will not be repeated here.

**5.4. Selection**

The selection operator is used to choose chromosomes from the current population for a mating pool. In the literature, several selection techniques have been proposed. Of these the ranking selection, which is analytically shown to be a good choice in [33], is determined as the selection operator.

The BGA implements the ranking selection with roulette wheel sampling. The details of this implementation can be found in [18, 30] and thus will not be repeated here. Remember that there are both valid and invalid chromosomes in the population. The ranking selection with roulette wheel sampling is used for only the valid chromosomes. From a set of invalid chromosomes, at most  $N_{iv}$  invalid chromosomes will be randomly selected for the mating pool, where each one has an equal chance of being selected. As a result, in the mating pool, there will be at least  $N_p - N_{iv}$  valid chromosomes selected from the valid chromosomes through the ranking selection with roulette wheel sampling and at most  $N_{iv}$  invalid chromosomes randomly selected from the invalid chromosomes. Note that if the number of invalid chromosomes is less than  $N_{iv}$ , valid chromosomes substitute for invalid ones. In addition, it is possible for a valid chromosome to appear more than once in the mating pool. After the selection is over, chromosomes in the mating pool will be subject to crossover and mutation operators to form the next generation.

**5.5. Crossover**

In the BGA, 1-point crossover [28] is implemented as follows. For each chromosome  $\mathcal{C}_i$ , it is possible to construct a list  $L_i$

into which scheduling lists  $L_i^j$  are combined. The following definition will be used in the procedure that combines the scheduling lists.

**DEFINITION 4.** *A valid position for a task in a list of tasks is defined to be a position such that if the task is placed at that position, there exist no predecessors (successors) of that task after (before) it in the list.*

All predecessor and successor tasks of a task can be found by using the depth-first search algorithm [29] before the search begins. Thus, it is possible to check the precedence relationship between any two tasks in constant time during the search.

The procedure for combining the scheduling lists of chromosome  $C_i$  into list  $L_i$  runs as follows. (i) Let  $L_i$  be an empty list. (ii) For each non-empty scheduling list  $L_i^j$ , starting from the end of list  $L_i^j$ , each task in  $L_i^j$  is placed at a position in  $L_i$  so that the execution order of tasks dictated by  $L_i^j$  is preserved and the position is the last available valid position. For at least one task of an invalid chromosome, finding a valid position in list  $L_i$  that preserves the execution order of the task due to  $L_i^j$  is impossible. In such a case, the task is placed to preserve its execution order. If the task is the last task of  $L_i^j$ , it is put at the end of the list. An important property of this procedure is that it produces a topologically sorted list for a valid chromosome.

The crossover operation is implemented as follows. (i) Chromosomes in the mating pool are randomly paired. (ii) A pair of non-crossed over chromosomes, e.g.  $C_i$  and  $C_j$ , is taken from the mating pool. (iii) The crossover operator is applied to the chosen pair with probability  $\mu_c$ , which is the probability of crossover and is experimentally determined, provided  $C_i$  and  $C_j$  are different. (iv) The procedure presented above is used to obtain lists  $L_i$  and  $L_j$ . (v) For the pair, a cut-off point, which divides lists  $L_i$  and  $L_j$  into top and bottom parts, is randomly generated. (vi) The machine to which task  $v_k$  in the bottom part of  $L_i$  ( $L_j$ ) is assigned is changed to the machine to which task  $v_k$  is assigned in list  $L_j$  ( $L_i$ ). (vii) Two new chromosomes are generated by assigning the tasks to the respective machines in the order dictated by lists  $L_i$  and  $L_j$  and these two new chromosomes are put back into the mating pool. Note that either or both of the newly generated chromosomes might be invalid.

### 5.6. Mutation

The mutation operator is implemented as follows. (i) A non-mutated chromosome is randomly chosen from the mating pool. (ii) The mutation operator is applied to the chosen chromosome with probability  $\mu_m$ , which is the probability of mutation and is experimentally determined. (iii) A task and a machine on which the task will be scheduled are randomly determined. (iv) This task is randomly placed at one of the valid positions in the scheduling list of the chosen machine. After the mutation, the chromosome is put back into the mating pool. Note that the mutated chromosome might be invalid.

After the mutation operator is applied to all chromosomes in the mating pool, the chromosomes in the mating pool constitute the next generation.

## 6. EXPERIMENTS

In order to evaluate the BDLs algorithm and the BGA and compare their performance with RDLs algorithm [20], a simulation program that can be used to emulate the execution of randomly generated or real application task graphs on a simulated computing system was developed.

In the simulation program, a heterogeneous computing system is created based on two parameters, namely the number of computation machines  $p$  and the number of communication machines  $q$ . Associated with each computation machine is a FIFO queue that holds the tasks scheduled on each particular machine. (Recall that each machine is assumed to execute a task in its queue to completion without preemption.) In order to interconnect computation machines, a network of  $q$  communication machines is employed, where the network topology is randomly generated and each computation machine is randomly connected to a communication machine. This simulation model closely mimics a computing system in which a set of machines is interconnected by a switched-based network. Other parameters of interest of the model are set as follows. The failure rates of machines and links are assumed to be uniformly distributed between  $10^{-3}$  and  $10^{-4}$  failures/h [7]; the transmission rates of links are assumed to be uniformly distributed between 1 and 10 Mbits/s.

The simulation studies performed are grouped into three sets: (i) executing randomly generated task graphs with different number of tasks (between 20 and 100) on a computing system with  $p = 20$  and  $q = 20$ , (ii) executing randomly generated task graphs with 50 tasks on a computing system with  $p$  ranging from 10 to 50 and  $q = 20$  and (iii) executing a real application task graph on a computing system with  $p = 50$  and  $q = 20$ ,  $p = 100$  and  $q = 40$ ,  $p = 150$  and  $q = 60$ ,  $p = 200$  and  $q = 80$ , and  $p = 250$  and  $q = 100$ . For the randomly generated task graphs, the execution time of each task of the task graph is assumed to be uniformly distributed between 10 and 120 min, where the execution times of a given task are different on different machines. Furthermore, the volume of data to be transmitted among tasks is randomly generated such that the communication to computation ratio (CCR) is 1.0 or 10.0, where the average communication time between a task and its successor tasks is set to the average execution time of the task multiplied by the CCR. As far as the RDLs algorithm is concerned, it can assume one of the three cost functions developed in [20] for computing  $C(v_i, m_j)$ . In this study, the RDLs with the first cost function is assumed since it leads to a middling performance compared with the RDLs algorithms with the second and third cost functions according to the simulation results in [20].

In the simulations, the execution time and failure probability of applications were measured for the different

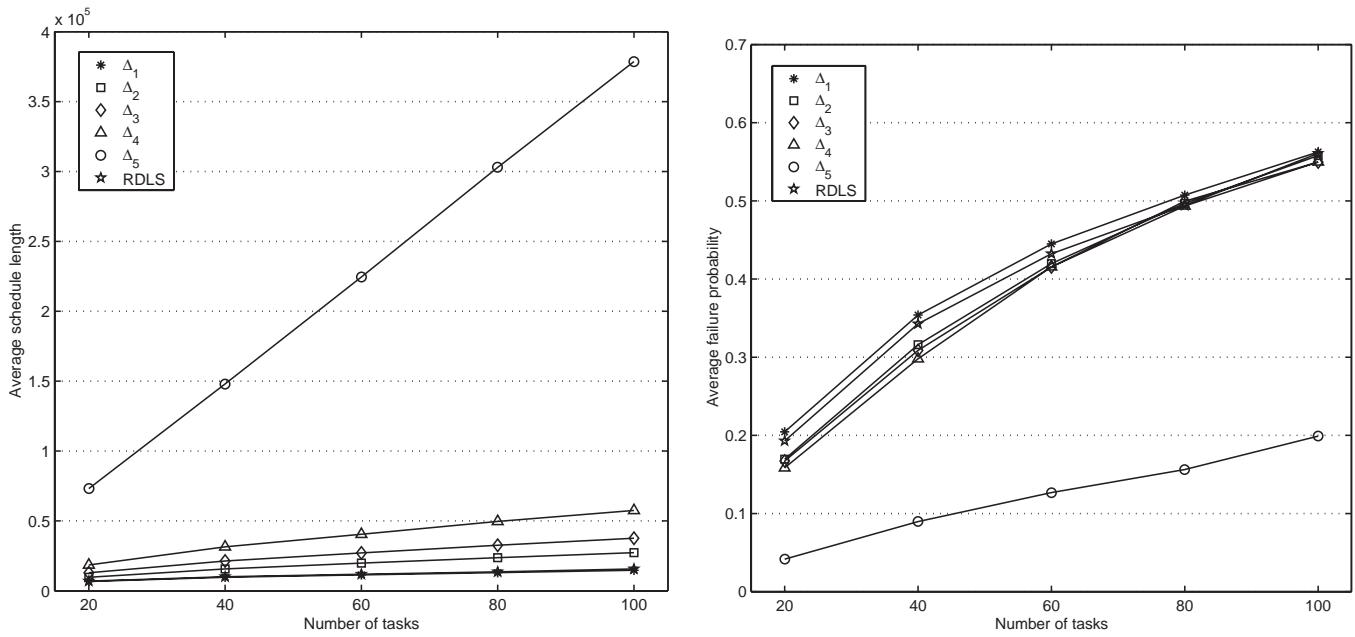


FIGURE 4. Average schedule length and failure probability of applications with CCR = 1.0 under the BDLs algorithm.

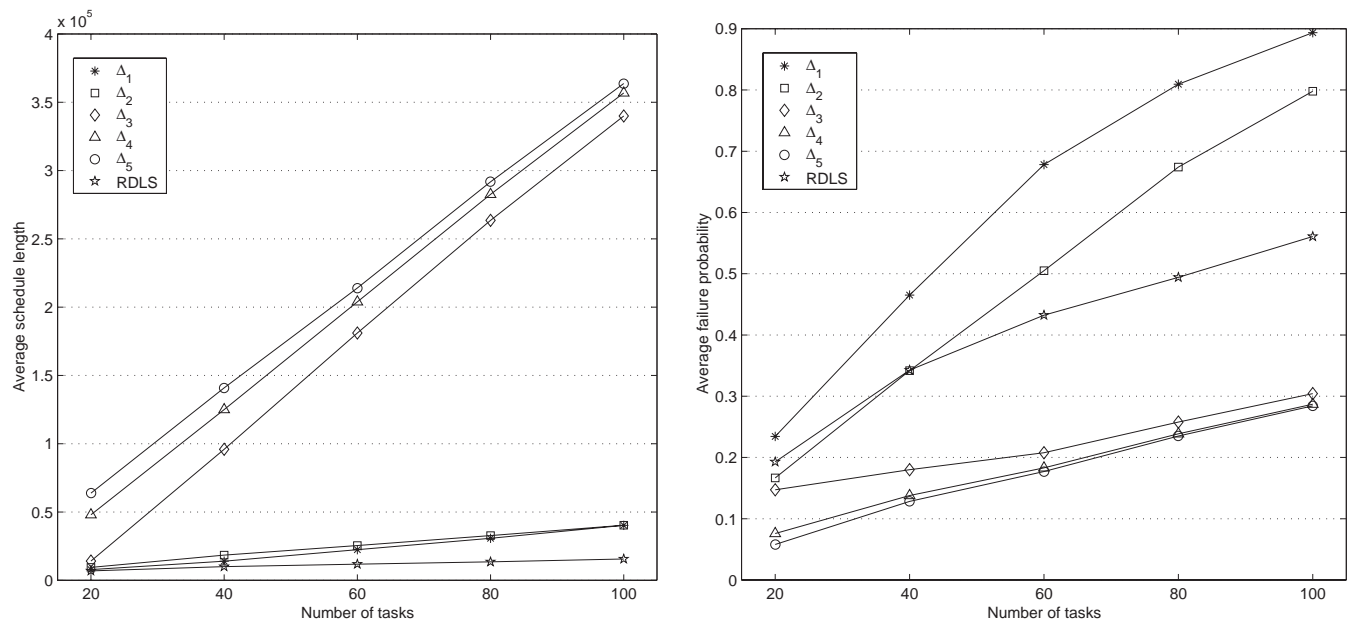


FIGURE 5. Average schedule length and failure probability of applications with CCR = 1.0 under the BGA.

values of  $\delta^1$  and  $\delta^2$ . Let  $\Delta_i = [\delta^1, \delta^2]$ ;  $\Delta_1 = [1, 0]$ ,  $\Delta_2 = [0.75, 0.25]$ ,  $\Delta_3 = [0.5, 0.5]$ ,  $\Delta_4 = [0.25, 0.75]$  and  $\Delta_5 = [0, 1]$ . Note that the weight for reliability increases as  $i$  gets larger. The parameters of the BGA were set as  $N_p = 100$ ,  $N_{iv} = 5$ ,  $\mu_c = 0.65$ ,  $\mu_m = 0.1$  and  $A = 1.1$ . The BGA was stopped if the quality of the elite chromosome did not improve over 30 generations.

The simulation results are shown in Figures 4–11. Specifically, Figures 4 and 7 present the results of the first set of simulation studies. According to Figures 4 and 7, the failure probability of an application (random task graph)

increases in proportion to the size of the application. This is due to the fact that when the size of an application increases, computation machines have to be failure-free for longer time periods for the execution of tasks and communication resources have to be failure-free for longer time periods for the transmission of intertask data items. Since the failure probability of a resource increases exponentially within the time interval for which the resource must remain failure-free, the failure probability of the application increases. As a result, a large, long running application will be more susceptible to failures, unless tasks of the application are

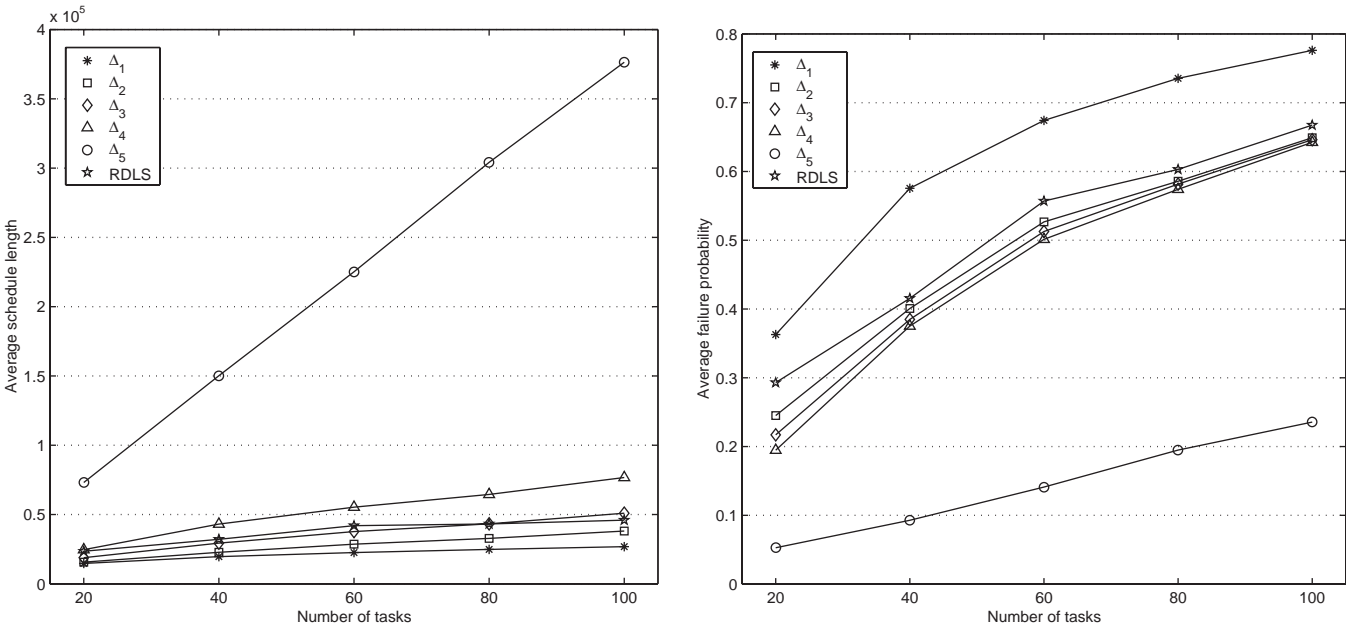


FIGURE 6. Average schedule length and failure probability of applications with CCR = 10.0 under the BDLs algorithm.

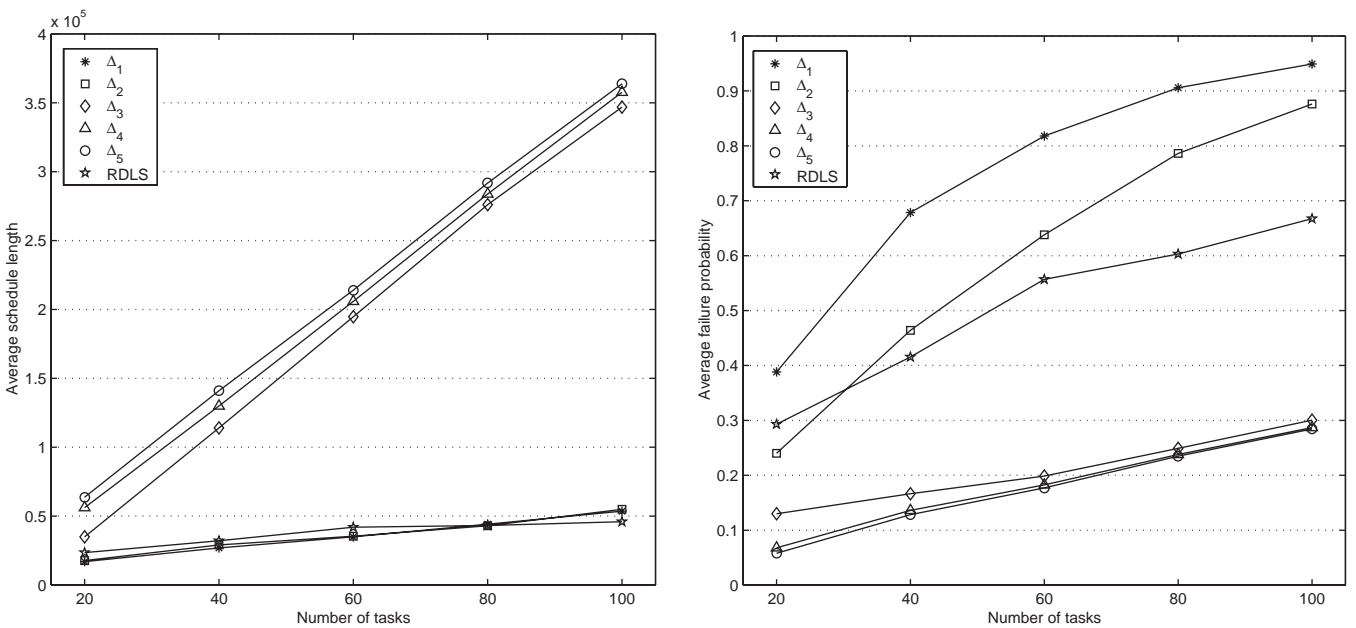


FIGURE 7. Average schedule length and failure probability of applications with CCR = 10.0 under the BGA.

assigned to machines in such a way that the reliability of the resources committed for the execution of the application is accounted for.

The ability of the BDLs algorithm and the BGA to trade off execution time against reliability is clear in Figures 4–11. These figures show that (i) the average schedule length is lowest when  $\Delta_1 = [1, 0]$  and highest when  $\Delta_5 = [0, 1]$ ; (ii) while  $\delta^1$  decreases and  $\delta^2$  increases the average schedule length and reliability of an application increase and vice versa; (iii) the failure probability is largest when  $\Delta_1 = [1, 0]$  and lowest when  $\Delta_5 = [0, 1]$ . It should be noted that

since there are no assigned capacity limits for computation machines, such as on processing time and memory capacity, both the BDLs algorithm and the BGA tend to schedule all the tasks of an application to a few of the most reliable machines for  $\Delta_5 = [0, 1]$ . As a result, the failure probability becomes lowest but the schedule length increases dramatically.

The results of the second simulation studies are shown in Figures 8 and 9, where the random task graphs with CCR = 1.0 are assumed. In Figure 8, while the number of computation machines increases, the BDLs algorithm consistently leads to a decreased schedule length except

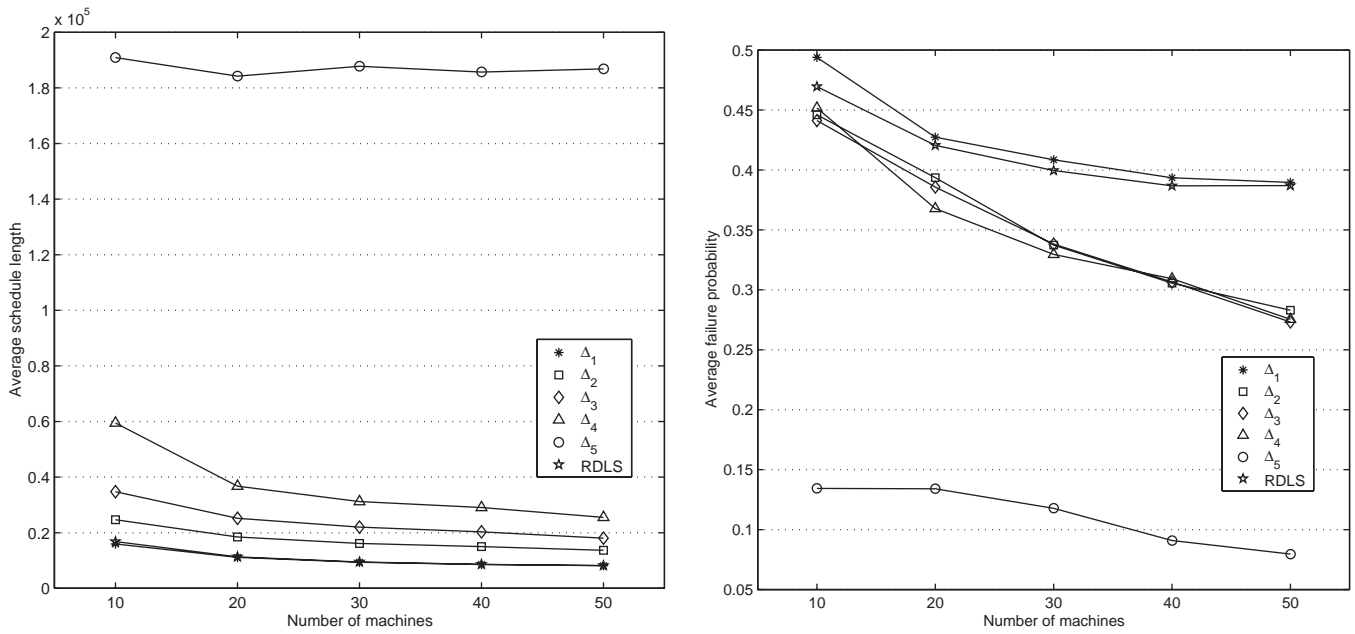


FIGURE 8. Average schedule length and failure probability of applications with  $CCR = 1.0$  under the BDLS algorithm with respect to an increasing number of machines.

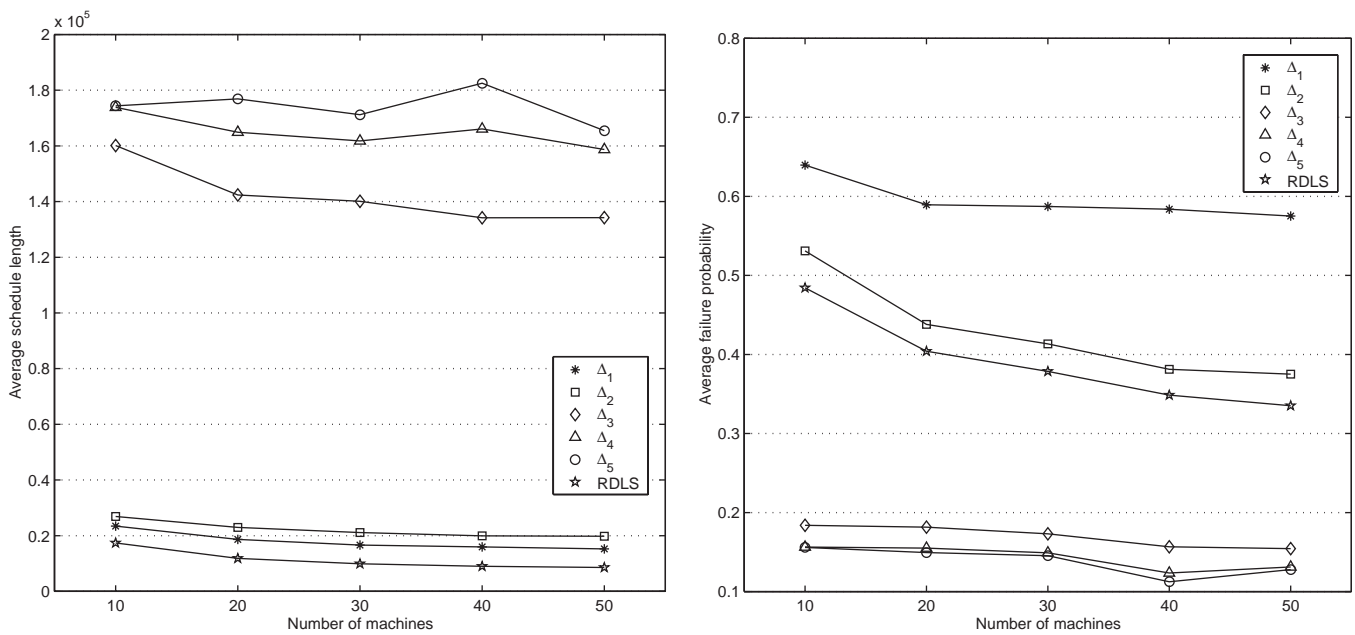


FIGURE 9. Average schedule length and failure probability of applications with  $CCR = 1.0$  under the BGA with respect to an increasing number of machines.

for  $\Delta_5$ , as expected, which also results in a decreased failure probability. In Figure 9, however, the BGA does not show as much consistency as the BDLS algorithm. This may be attributed to the fact that the BGA is a stochastic search technique, whereas the BDLS algorithm is a deterministic one.

For the third set of simulation studies, a real application (CSTEM) task graph shown in Figure 12

with  $CCR = 1.0$  is used. CSTEM (Coupled Structural–Thermal–Electromagnetic Analysis and Tailoring of Graded Composite Structures) [34] is a finite element-based computer program. The numbers within the nodes of the graph in Figure 12 represent the approximate execution times, in seconds, on a Sun Microsystems Sparc 10 workstation [34]. Since the execution time of each task depends on the size of the problem, the task execution times specified in the task

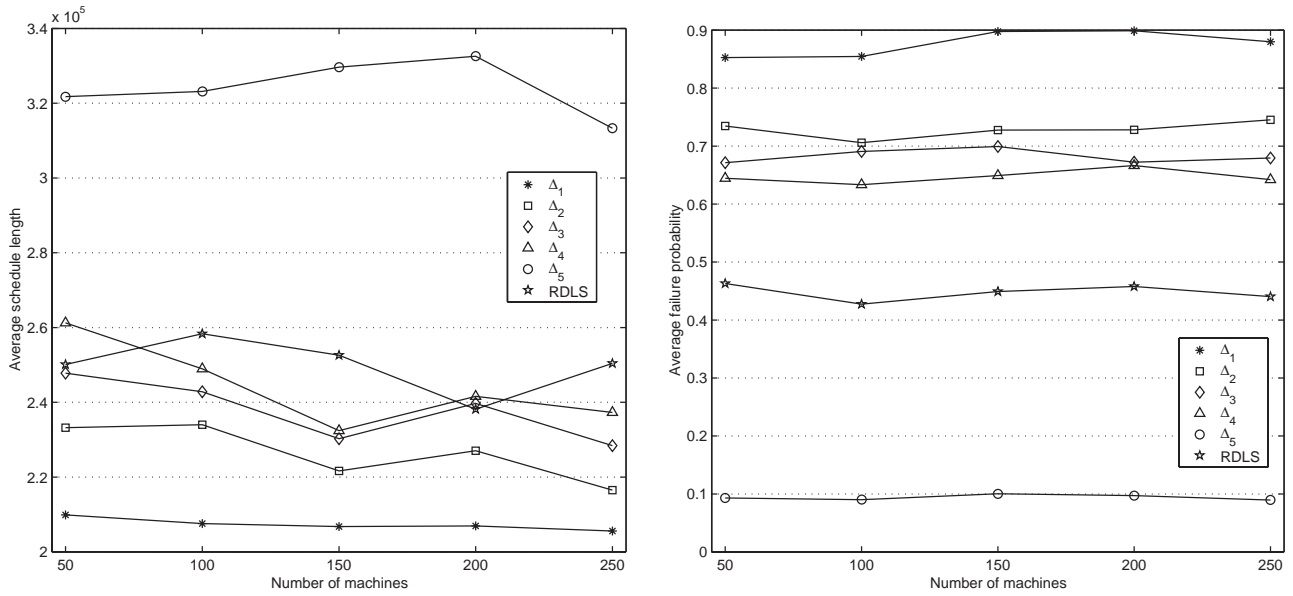


FIGURE 10. Average schedule length and failure probability of the CSTEM with CCR = 1.0 under the BDLs algorithm with respect to an increasing number of machines.

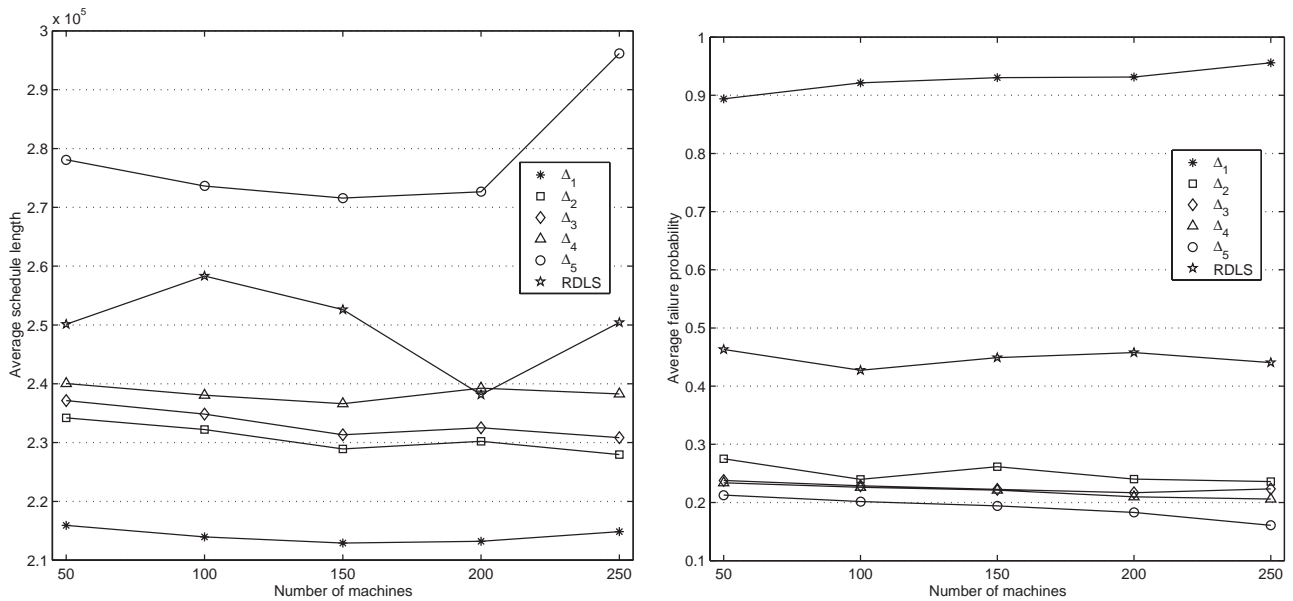


FIGURE 11. Average schedule length and failure probability of the CSTEM with CCR = 1.0 under the BGA with respect to an increasing number of machines.

graph are changed to be used in the simulations as follows. The task with the smallest execution time (0.3 s) is assumed to take 10 min on average. Then, the average execution times (in seconds) of the other tasks are found by multiplying their execution times given in the task graph by  $(10 \times 60)/0.3$ . Under these settings, the third set of simulation studies is shown in Figures 10 and 11, which are similar to the results of the second set of simulation studies.

For all simulation results and  $\Delta_1$ , the BGA produces dominated task assignments compared with the BDLs algorithm. Note that a task assignment is said to be dominated

by another one if and only if it leads to both higher schedule length and higher failure probability. Except for the third set of simulation results, for  $\Delta_2$ , the BGA once again is mostly dominated by the BDLs algorithm. For the third set of simulation results and  $\Delta_3$  and  $\Delta_4$ , the BGA is the non-dominated one. For the rest of the simulation studies and  $\Delta_2$ ,  $\Delta_3$ ,  $\Delta_4$  and  $\Delta_5$ , both the BDLs algorithm and the BGA are non-dominated between them.

As far as the performance of the RDLS algorithm is concerned, remember that the BDLs algorithm turns into the DLS algorithm for  $\Delta_1$ . When the figures for the BDLs

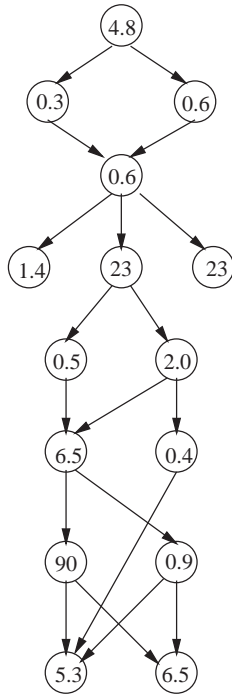


FIGURE 12. The task graph of CSTEM.

algorithm are studied, it is clear that the RDLS algorithm has not perturbed the DLS algorithm, i.e. the schedule length under the BDLS algorithm with  $\Delta_1$  is close to that under the RDLS algorithm, for the first set of simulations with  $CCR = 1.0$  and the second set of simulations. This is because the term  $DL(v_i, m_j)$  mostly dominates the term  $C(v_i, m_j)$  in the objective function of the RDLS algorithm for  $CCR = 1.0$ . When  $CCR = 10.0$ ,  $C(v_i, m_j)$  becomes comparable to  $DL(v_i, m_j)$  and affects the scheduling decisions more. Thus, it is safe to say that the RDLS algorithm performs somewhere between the BDLS algorithm with  $\Delta_1$  and the BDLS algorithm with  $\Delta_2$  for  $CCR = 1.0$  for the aforementioned simulation cases. This is also true for small values of  $CCR$  other than  $CCR = 1.0$ . For  $CCR = 10.0$ , the BDLS algorithm with  $\Delta_1$ ,  $\Delta_2$  and  $\Delta_3$  mostly dominates the RDLS algorithm, which generally dominates the BGA with  $\Delta_1$  and  $\Delta_2$ . For the third set, the RDLS algorithm dominates the BGA with  $\Delta_2$ ,  $\Delta_3$  and  $\Delta_4$ . In the rest of the simulation studies, the BDLS and RDLS algorithms and the BGA and the RDLS algorithms are non-dominated among themselves.

Our simulation program is run on a PC with the following specifications: Intel Pentium 4 processor with 1.6 GHz clock speed, 128 MBytes of RAM and Red Hat 8.0 Linux operating system. The running times of the BDLS algorithm and the BGA are measured as follows. For the first set of simulation studies, the BDLS algorithm's running times with an increasing number of tasks are 0.11, 0.92, 3.33, 11.79 and 21.44 sec; the BGA's running times are 2.34, 7.35, 11.27, 17.06 and 26.44 sec. For the second set of simulation studies, the BDLS algorithm's running times with an increasing number of machines are 1.09, 2.12, 3.53, 3.93 and 5.05 sec;

the BGA's running times are 7.35, 8.70, 9.03, 11.17 and 11.53 sec. For the simulation of the CSTEM executing on relatively large HC systems, the BDLS algorithm's running times are 0.09, 0.29, 0.50, 0.87 and 1.19 sec; the BGA's running times are 0.97, 1.87, 2.00, 2.80 and 2.90 sec. As expected, the running times of both algorithms increase with an increasing number of tasks and machines. Furthermore, the BDLS algorithm runs faster than the BGA.

### 7. CONCLUSIONS

In this paper, two algorithms are developed for matching and scheduling a DAG-structured application with the goal of minimizing execution time and failure probability of the application. Since it is not usually possible to achieve these two conflicting objectives at the same time, both algorithms are designed to trade off execution time against the failure probability of the application. The BDLS algorithm is obtained by modifying an existing scheduling algorithm, the DLS algorithm. In a similar manner, the steps taken to transform the DLS algorithm into the BDLS algorithm may be followed to have other existing scheduling algorithms account for the reliability of resources in making scheduling decisions. The BGA is inspired by the fact that GAs are very effective in producing Pareto-optimal solutions for multiobjective optimization problems. Thus, the BGA can be used not only for producing non-dominated task assignments but also as a benchmark algorithm against which the performance of new algorithms can be compared.

The simulation studies in the previous section showed that both the BDLS algorithm and the BGA are capable of trading execution time for the failure probability of applications. As a result, both algorithms can attain both objectives to some degree by producing compromise task assignments. Furthermore, having a compromise task assignment is important in that it may be required for a large, long running application to limit its execution time with relatively low failure probability.

According to the simulation results, there is no clear winner between the BDLS algorithm and the BGA. A disadvantage of the BGA is that it is slow compared with the BDLS. If the problem size is big, i.e. the numbers of machines and tasks are in the order of thousands, it will take a long time for the BGA to converge. On the other hand, even for such big problems, the BDLS algorithm can be used to produce solutions with a reasonable degree of quality in a relatively short time period. The solutions generated by the BDLS algorithm can be used by the BGA as initial solutions to come up with non-dominated task assignments meeting the desired trade-off.

Finally, a new mathematical model is proposed to compute the failure probability of a DAG-structured application executing on a HC system. As discussed, the proposed model provides a more accurate estimation of the failure probability than the models presented in the literature. In addition, unique to the model is that it is not restricted to tree network topologies.

## REFERENCES

- [1] Eshagian, M. M. (1996) *Heterogeneous Computing*. Artech House, Norwood.
- [2] Foster, I. and Kesselman, C. (1997) Globus: a metacomputing infrastructure toolkit. *Int. J. Supercomput. Ap.*, **11**, 115–128.
- [3] Sih, G. C. and Lee, E. A. (1993) A compile-time scheduling heuristic for interconection-constraint heterogeneous processor architectures. *IEEE Trans. Parall. Distr.*, **4**, 175–187.
- [4] Iverson, M. A. and Özgüner, F. (1998) Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment. *IPPS/SPDP Workshop on Heterogeneous Computing*, Orlando, FL, March 30, pp. 70–78. IEEE Computer Society, Los Alamitos.
- [5] Carter, B. R., Watson, D. W., Freund, R. F., Keith, E., Mirabile, F. and Siegel, H. J. (1998) Generational scheduling for dynamic task management in heterogeneous computing systems. *Inform. Sciences*, **106**, 219–236.
- [6] Maheswaran, M. and Siegel, H. J. (1998) A dynamic matching and scheduling algorithm for heterogeneous computing systems. *IPPS/SPDP Workshop on Heterogeneous Computing*, Orlando, FL, March 30, pp. 57–69. IEEE Computer Society, Los Alamitos.
- [7] Plank, J. S. and Elwasif, W. R. (1998) Experimental assessment of workstation failures and their impact on checkpointing systems. In *Proc. Int. Symp. on Fault-Tolerant Computing*, Munich, Germany, June 23–25, pp. 48–57. IEEE Computer Society, Los Alamitos.
- [8] Kumar, A., Rai, S. and Agrawal, D. P. (1988) On computer communication network reliability under program execution. *IEEE J. Sel. Area Comm.*, **6**, 1393–1399.
- [9] Hwang, G.-J. and Tseng, S.-S. (1993) A heuristic task-assignment algorithm to maximize reliability of a distributed system. *IEEE Trans. Reliab.*, **42**, 408–415.
- [10] Chang, M.-S., Chen, D.-J. and Ku, K.-L. (1999) The distributed program reliability analysis on a star topology: efficient algorithms and approximate solution. *IEICS Trans. Information and Systems*, **E82-D**, 1020–1029.
- [11] Shatz, S. M., Wang, J. P. and Goto, M. (1992) Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Comput.*, **41**, 1156–1168.
- [12] Shatz, S. M. and Wang, J. P. (1989) Models & algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Trans. Reliab.*, **38**, 16–26.
- [13] Kartik, S. and Murthy, C. S. R. (1997) Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Trans. Comput.*, **46**, 719–724.
- [14] Srinivasan, S. and Jha, N. K. (1999) Safety and reliability driven task allocation in distributed systems. *IEEE Trans. Parall. Distr.*, **10**, 238–251.
- [15] Doğan, A. and Özgüner, F. (2000) Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous computing. In *Proc. ICCP Workshop on Network Based Computing*, Toronto, Canada, August 21–24, pp. 429–436. IEEE Computer Society, Los Alamitos.
- [16] Qin, X. and Jiang, H. (2001) Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In *Proc. Int. Conf. Parallel Processing*, Valencia, Spain, September 3–7, pp. 113–122. IEEE Computer Society, Los Alamitos.
- [17] Hou, C.-J. and Shin, K. G. (1997) Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Trans. Comput.*, **46**, 1338–1356.
- [18] Doğan, A. (2001) Matching and Scheduling of Applications in Heterogeneous Computing Systems with Emphasis on High-Performance, Reliability, and QoS. PhD thesis, The Ohio State University, Columbus, OH.
- [19] Iverson, M. A. (1999) Dynamic Mapping and Scheduling Algorithms for a Multi-User Heterogeneous Computing Environment. PhD thesis, The Ohio State University, Columbus, OH.
- [20] Doğan, A. and Özgüner, F. (2002) Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. Parall. Distr.*, **13**, 308–323.
- [21] Yang, J., Khokhar, A., Sheikh, S. and Ghafoor, A. (1994) Estimating execution time for parallel tasks in heterogeneous processing (HP) environment. In *Proc. IPPS Workshop on Heterogeneous Computing*, Cancún, Mexico, April 26, pp. 23–28. IEEE Computer Society, Los Alamitos.
- [22] Iverson, M. A., Özgüner, F. and Potter, L. (1999) Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. *IEEE Trans. Comput.*, **48**, 1374–1379.
- [23] Coello, C. A. C. (1999) A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, **1**, 269–308.
- [24] Lewis, E. E. (1987) *Introduction to Reliability Engineering*. John Wiley & Sons.
- [25] Patvardhan, C., Prasad, V. C. and Pyara, V. P. (1997) Generation of K-trees of undirected graphs. *IEEE Trans. Reliab.*, **46**, 208–211.
- [26] Ball, M. O. (1986) Computational complexity of network reliability analysis: an overview. *IEEE Trans. Reliab.*, **R-35**, 230–239.
- [27] Doğan, A. and Özgüner, F. (2003) *Biobjective Scheduling Algorithms for Execution Time - Reliability Trade-off in Heterogeneous Computing Systems*. Technical Report 2003-001, Department of Electrical and Electronics Engineering, Anadolu University, Eskişehir, Turkey.
- [28] Beasley, D., Bull, D. R. and Martin, R. R. (1993) An overview of genetic algorithms: part 1, fundamentals. *U. Comput.*, **15**, 58–69.
- [29] Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1997) *Introduction to Algorithms*. MIT Press, Cambridge.
- [30] Wang, L., Siegel, H. J., Roychowdhury, V. P. and Maciejewski, A. A. (1997) Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *J. Parallel Distr. Comput.*, **47**, 8–22.
- [31] Bentley, P. J. and Wakefield, J. P. (1996) *An Analysis of Multi-objective Optimization within Genetic Algorithms*. Technical Report ENGPJB96, University of Huddersfield, UK.
- [32] Rudolph, G. (1994) Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networ.*, **5**, 96–101.
- [33] Goldberg, D. E. and Deb, K. (1991) A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, pp. 69–93. Morgan Kaufmann Publishers, Inc.
- [34] Iverson, M. A., Özgüner, F. and Follen, G. J. (1995) Parallelizing existing applications in a distributed heterogeneous environment. In *Proc. IPPS Workshop on Heterogeneous Computing*, Santa Barbara, CA, April 24, pp. 93–100. IEEE Computer Society, Los Alamitos.