

Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing

Atakan Dogan, *Student Member, IEEE*, and Füsün Özgüner, *Member, IEEE*

Abstract—In a heterogeneous distributed computing system, machine and network failures are inevitable and can have an adverse effect on applications executing on the system. To reduce the effect of failures on an application executing on a failure-prone system, matching and scheduling algorithms which minimize not only the execution time but also the probability of failure of the application must be devised. However, because of the conflicting requirements, it is not possible to minimize both of the objectives 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 reliability of the application. This goal is achieved by modifying an existing matching and scheduling algorithm. The reliability of resources is taken into account using an incremental cost function proposed in this paper and the new algorithm is referred to as the *reliable dynamic level scheduling algorithm*. The incremental cost function can be defined based on one of the three cost functions developed here. These cost functions are unique in the sense that they are not restricted to tree-based networks and a specific matching and scheduling algorithm. The simulation results confirm that the proposed incremental cost function can be incorporated into matching and scheduling algorithms to produce schedules where the effect of failures of machines and network resources on the execution of the application is reduced and the execution time of the application is minimized as well.

Index Terms—Matching and scheduling, precedence-constrained tasks, heterogeneous computing, reliability, articulation points and bridges, DLS algorithm.



1 INTRODUCTION

A heterogeneous distributed computing system (HDCS) is a suite of diverse high-performance machines interconnected by a high-speed network, thereby promising high-speed processing of computationally intensive applications with diverse computing needs. One of the challenges in heterogeneous computing is to develop matching and scheduling algorithms that assign the tasks of the application to the machines [1], [2]. Many static, dynamic, and even hybrid algorithms have been proposed to minimize the execution time of applications running on a heterogeneous computing system [3], [4], [5], [6], [7]. In such a large network of machines, machine and network failures are inevitable and can have an adverse effect on applications executing on the system. One way of taking failures into account is to employ a reliable matching and scheduling algorithm in which tasks of an application are assigned to machines to minimize the probability of failure of that application. This type of approach was followed for allocating undirected task graphs to nonredundant and redundant real-time distributed systems [8], [9], [10], [11]. Recently, algorithms for reliable matching and scheduling

of tasks with precedence constraints on HDCSs were also developed [12], [13]. However, since the only objective of a reliable matching and scheduling algorithm is to minimize the probability of failure of the application, it may produce task assignments that increase the execution time of the application. As a result, there are usually conflicting requirements between minimizing the execution time and the probability of failure of an application, and it may not be possible to minimize both objectives at the same time. Consequently, matching and scheduling algorithms which take into account both execution time and reliability must be devised. A method developed in this manner for tree-based networks was presented in [14], where a failure cost function was incorporated into an existing matching and scheduling algorithm to take failures into account while making matching and scheduling decisions.

The methods presented in [8], [9], [10], [11], [12], [13], [14] are restricted to tree-based networks due to the fact that the computation of the probability that the communication between two machines in a network is failure-free (reliability of the communication) is *NP*-hard [15]. Several methods have been proposed to compute this probability for general networks [16], [17], [18], [19]. However, these methods cannot be included in a matching and scheduling algorithm because of their high complexities. To develop a matching and scheduling algorithm which works for an arbitrary network topology, a method of approximating this probability must be determined.

• The authors are with the Department of Electrical Engineering, The Ohio State University, 2015 Neil Av., Columbus, OH 43210-1272.
E-mail: {dogana, ozguner}@ee.eng.ohio-state.edu.

Manuscript received 11 July 2000; revised 24 Mar. 2001; accepted 10 Sept. 2001.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 112444.

The algorithm presented in this paper for matching and scheduling of tasks with precedence constraints is the first in the sense that it is not restricted to tree-based networks as in the algorithms proposed in [8], [9], [10], [11], [12], [13], [14]. This is because of the method presented in Section 5 which can be used to estimate the reliability of the communication between two machines. Furthermore, while algorithms presented in [3], [4], [5], [6], [7] and [8], [9], [10], [11], [12], [13] deal with only one objective, which is minimizing the execution time and minimizing the probability of failure, respectively, the algorithm presented here takes both of the objectives into account. To account for the first objective, the cost function of an existing compile time, static list scheduling heuristic [3] is used. To account for the second objective, three *cost functions*, which can associate a cost in time units with a matching and scheduling decision, are developed. Based on these cost functions, the effect of a matching and scheduling decision on the unreliability of the application is defined as a cost in time units, referred to as the *incremental cost function*. The fact that many matching and scheduling algorithms express their cost functions in terms of time units allows the incremental cost function to be compatible with many static or dynamic scheduling algorithms. Finally, the function given in [3] and the incremental cost function developed in this paper are combined into one cost function to form a new matching and scheduling algorithm, which is called the *reliable dynamic level scheduling algorithm*. The simulation results in Section 7 show that this new algorithm can produce task assignments that minimize both the execution time and the probability of failure of the application.

The rest of the paper is organized as follows: Section 2 gives preliminaries and Section 3 presents the *reliable dynamic level scheduling algorithm*. Section 4 gives some definitions and a theorem which are used in designing the cost functions. Section 5 introduces the approximation method used for estimating the reliability of the communication and Section 6 presents the cost functions which are used to associate a cost with each matching and scheduling decision and the incremental cost function. Section 7 evaluates the effects of the proposed incremental cost function on the list scheduling heuristic and Section 8 gives our concluding remarks.

2 PRELIMINARIES

2.1 Network and Application Models and Assumptions

The network topology is represented by a connected, undirected graph $G = (M, N)$, where set M denotes heterogeneous machines and set N denotes communication links. Let $m_j \in M$ denote a machine, where $1 \leq j \leq p$ and $n_{k,l} \in N$ denote the link between machines m_k and m_l . $n_{k,l}$ and $n_{l,k}$ refer to the same link. Note that this model of the network assumes no Ethernet type of network connections among machines, which is the only limitation of the model. Let $\mathcal{R} = M \cup N$ denote the set of resources in the HDCS; element $r_i \in \mathcal{R}$ refers to either a machine or a network link. This set is introduced for only notational convenience. A *simple path* between two machines m_s and m_t is defined to be a set of resources that form a path from m_s to m_t in which a resource does not appear more than once. That

resource set includes both the source and destination machines as well.

An application executing on the system is represented using a directed acyclic task graph (DAG) $T = (V, E)$, where set V denotes tasks to be executed and set E denotes the communication among tasks. Let $v_i \in V$ denote a task of the application, where $1 \leq i \leq n$ and $e_{i,j} \in E$ denote the communication between tasks v_i and v_j , where task v_i is said to be an immediate predecessor of task v_j . A source machine $m_{src} \in M$, which represents the location of the user in the network, is also associated with the application. This machine supplies all initial data needed by the application. Furthermore, all results produced by the application must be transmitted to this machine. Thus, new vertices, namely, v_s and v_e , and edges are added to task graph T to model these data dependencies. The new DAG obtained is represented by $T' = (V', E')$. In task graph T' , v_s is the only source task, from which an edge is drawn to each source task of task graph T , and v_e is the only exit task, to which an edge is drawn from each exit task of task graph T . Both tasks v_s and v_e have zero computation costs and are always scheduled on the source machine of the application (m_{src}). As a result, all source tasks in T will receive data from the source task v_s and all exit tasks in T will send results to exit task v_e .

The failure of a resource in the system is assumed to follow a Poisson process and each resource $r_i \in \mathcal{R}$ is associated with a constant failure rate λ_{r_i} accordingly. 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 experimentally shown in [20] that such an assumption may 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. It should be emphasized that these three assumptions about failures are common to other studies which deal with analyzing the reliability of computer systems [8], [9], [10], [11], [12], [13], [14], [17].

2.2 Articulation Points, Bridges, and Biconnected Components

The following definitions are adapted from the original definitions given in [21].

Definition 1. An articulation point of graph G is a vertex (machine) whose failure disconnects the graph.

Definition 2. A bridge of graph G is an edge (link) whose failure disconnects the graph.

Definition 3. A biconnected component of graph G is a subgraph which does not have an articulation point.

Fig. 1 illustrates these definitions. Articulation points, bridges, and biconnected components of a graph can be found using the *depth-first search* algorithm in $O(|M| + |N|)$ time [21]. In Section 5, it is shown that the reliability of the communication between machines m_s and m_t in an arbitrary network can be computed by using the articulation points and bridges along the shortest path between m_s and m_t .

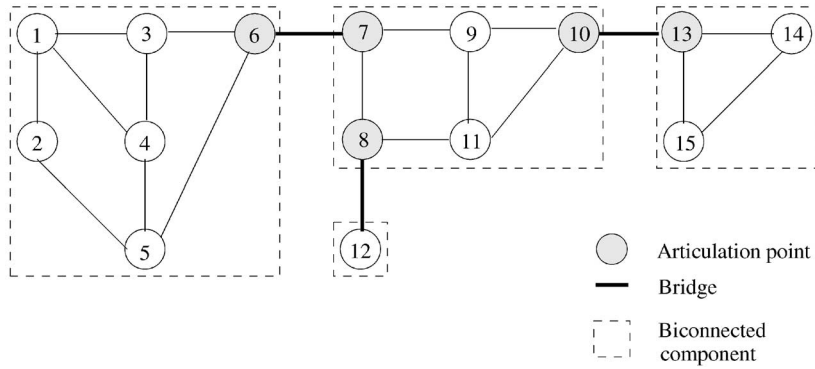


Fig. 1. Articulation points, bridges, and biconnected components of a connected, undirected graph.

3 SCHEDULING ALGORITHMS

In this section, an existing matching and scheduling algorithm is first presented. After that, this algorithm is modified to account for the reliability of resources while scheduling tasks of a directed acyclic task graph.

3.1 DLS Algorithm

The *dynamic level scheduling* (DLS) algorithm is a compile time, static list scheduling heuristic which 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 have the highest dynamic level. The *dynamic level* of a task-machine pair, $DL(v_i, m_j)$, is defined to be:

$$DL(v_i, m_j) = SL(v_i) - \max\{t_{i,j}^A, t_j^M\} + \Delta(v_i, m_j). \quad (1)$$

The first term in (1) is called the static level of the task. The *static level* of task v_i is defined to be the largest sum of the median execution times of the tasks along any directed path from task v_i to v_e . The static level indicates the importance of the task in the precedence hierarchy by giving higher priority to tasks from which the time spent to complete the execution of the application is expected to be larger. The max term defines the time when task v_i can begin execution on machine m_j , where $t_{i,j}^A$ denotes the time when the data will be available if task v_i is scheduled on machine m_j and t_j^M denotes the time when machine m_j will be available for the execution of task v_i . A task-machine pair with an earlier starting time will have higher scheduling priority. The third term accounts for the machine speed differences and is defined to be:

$$\Delta(v_i, m_j) = \bar{t}_i^E - t_{i,j}^E, \quad (2)$$

where \bar{t}_i^E denotes the median execution time of task v_i across all machines and $t_{i,j}^E$ denotes the execution time of task v_i on machine m_j . If machine m_j executes task v_i faster than the other machines in the network, Δ will be positive, which increases the scheduling priority.

3.2 RDLS Algorithm

The DLS algorithm does not account for the reliability of resources in the HDCS while making scheduling decisions. To address this problem, the *reliable dynamic level scheduling* (RDLS) algorithm is developed.

The terms in the cost function of the DLS algorithm collectively associate a priority in time units with each matching and scheduling decision for evaluating the effect of different decisions on the execution time of the application, where each term is defined in time units. In a similar manner, a new term can be incorporated into (1) to account for the reliability of the resources on which tasks will be executed. This new term must be defined in time units to be compatible with the other terms in (1). Consequently, in this study, a new term is incorporated into (1) as follows:

$$DL(v_i, m_j) = SL(v_i) - \max\{t_{i,j}^A, t_j^M\} + \Delta(v_i, m_j) - C(v_i, m_j). \quad (3)$$

This new algorithm will be referred to as the *reliable dynamic level scheduling* algorithm. In the RDLS algorithm, the first three terms that come from the DLS algorithm promote the best suited resources to the application to minimize the execution time of that application, while the last term $C(v_i, m_j)$ promotes resources with high reliability to maximize the reliability of the application. Similar to the DLS algorithm, the RDLS algorithm will seek a task-machine pair with the highest dynamic level given by (3). Since a large value of $C(v_i, m_j)$ will lower the scheduling priority, $C(v_i, m_j)$ can be thought as a measure of how much a given scheduling decision will contribute to the unreliability of the application. The rest of the paper is devoted to the computation of the $C(v_i, m_j)$ term.

4 TASK AND APPLICATION FAILURE

In this section, a theorem is presented that defines the time intervals in which a set of resources must remain failure-free in order for the application to successfully complete execution.

The set of resources that task v_i will rely on during its execution and the time interval associated with task v_i are given by the following definition.

Definition 4. A task $v_i \in V$ executing on machine m_j is defined to be failure-free if there exists a failure-free simple path between machines m_j and m_{src} during the time interval $[s_i, f_i]$.

The time interval $[s_i, f_i]$ is called the *duration of the execution*, where s_i denotes the time when the matching and scheduling decision is made for task v_i , and f_i denotes the time when task v_i has finished transmitting all of its results to its immediate successor tasks in V' . Since matching and scheduling decisions are made before the start of the execution of the application for a static matching and scheduling algorithm and these decisions cannot be changed afterwards, without loss of generality, s_i is set to zero for all tasks of the application. When task v_i is considered for scheduling, f_i cannot be computed due to the fact that the successor tasks of task v_i have not been scheduled yet. Therefore, the communication time between task v_i and its successor task(s) is approximated as the average time required to send a message (with size equal to the total number of bytes sent to all successor tasks) to machine m_{src} from the other machines.

It should be noted that Definition 4 does not impose any conditions on how the communication between tasks should be done. As a result, the communication between two tasks will be handled in the usual manner. With respect to Definition 4, if some resources have failed during the execution of task v_i , task v_i can still be considered to be failure-free provided that there exists a failure-free path between machines m_j and m_{src} during the duration of execution of task v_i . Note that, since a path between machines m_j and m_{src} includes the endpoints m_j and m_{src} , machines m_j and m_{src} are also failure-free during the duration of execution of task v_i .

Definition 5. The failure probability of an application is defined to be the probability that the application will not be completed due to the failure of a machine on which a task of the application is executing or the failure of the communication between two communicating tasks.

The failure probability of a DAG-structured application can be computed using the mathematical model proposed in [22].

Definition 6 [14]. An application is defined to be failure-free if no resource failure either 1) prevents or stops any task from executing or 2) prevents any task from communicating results to either a subsequent task or to the user.

Since the execution of an application begins and ends with the source machine m_{src} , it is obvious that this machine must remain failure-free during the execution of the application. For the other resources of the network, Theorem 1 below determines when the failure of a resource can affect the execution of an application. It should be noted that this theorem is an adaptation of the one presented in [14] for tree-based networks to general network topologies.

Theorem 1. An application is failure-free if and only if every task of the application is failure-free.

Proof. *Necessary condition:* If the application is failure-free, then every task of the application is failure-free. The necessary condition is proven by contradiction.

Assume that the application is failure-free and task v_i of the application executing on machine m_j is not failure-free. Note that there exists at least one directed path from $v_i \in V'$ to v_e , according to the definition of task graph T' . Since, due to the assumption, v_i is not failure-free, there exists no failure-free paths between m_j and m_{src} during the time interval $[s_i, f_i]$ (Definition 4). Therefore, the failure of the communication between m_j and m_{src} has disconnected the network into at least two portions, one containing m_j and the other m_{src} . On the other hand, if the application is failure-free, by Definition 6, each task can communicate with its immediate successor task along the directed path starting at v_i and ending with v_e . However, since the network is disconnected, one of the following is true: 1) If v_i is an exit task in task graph T , by Definition 4, the data produced by v_i cannot be transmitted to v_e . 2) Let $v_i \rightsquigarrow v_k \rightsquigarrow v_e$ denote a directed path between v_i and v_e and v_k be the immediate predecessor task of v_e . Suppose that all tasks along directed path $v_i \rightsquigarrow v_k$ can communicate with their immediate successor tasks. This means that all machines on which tasks along $v_i \rightsquigarrow v_k$ are assigned are in the same portion of the network and m_{src} is in the other. Consequently, v_k cannot have a failure-free path to v_e . 3) Let $v_i \rightsquigarrow v_k \rightsquigarrow v_l \rightsquigarrow v_e$ denote a directed path and v_k be the immediate predecessor task of v_l . Suppose that all machines on which tasks along $v_i \rightsquigarrow v_k$ are assigned are in the same portion of the network and all machines on which tasks along $v_l \rightsquigarrow v_e$ are assigned are in the other. Since v_i cannot communicate with v_e , v_k cannot transmit its data to v_l . With respect to 1), 2), and 3), at least one task along a directed path between v_i and v_e cannot communicate with its successor task. Thus, the application cannot be failure-free, which is a contradiction.

Sufficient Condition: If every task of the application is failure-free, then the application is failure-free. The sufficient condition is also proven by contradiction.

Assume that every task of the application is failure-free, and the application is not failure-free. Because of the assumption, at least one of the following is true by Definition 6: 1) A machine failure has prevented a task from executing (machine m_j executing task v_i has failed). 2) A resource failure has prevented a task from communicating results to either one of its successor tasks or to the user (the communication between machines m_j , on which task $v_i \in T'$ is executing, and m_p , on which task v_i 's predecessor task v_k is executing, has failed and machines m_j and m_p are failure-free). If 1) is true, then according to Definition 4, task v_i cannot be failure-free, which is a contradiction. If 2) is true, then one of the following is true as well: 1) The communication between m_j and m_{src} has failed, 2) the communication between m_p and m_{src} has failed, or 3) both 1) and 2) are true. This follows from the fact that, if there exists a failure-free path between m_j and m_{src} and m_p and m_{src} , there must be a failure-free path between m_j and m_p as well. As a result, by Definition 4, v_i , v_k , or both cannot be failure-free, which is a contradiction. \square

Note that with respect to Definition 4, the reliability of the communication between two particular machines would be crucial for the reliability of a task. Therefore, a

way of computing the reliability of the communication between two machines is needed.

5 RELIABILITY COMPUTATION

The *2-terminal* or *terminal-pair* reliability problem, which is to find the probability that there exists an operating path from a source machine to a destination machine, is one of the basic reliability problems and its complexity is *NP-hard* [15]. Existing terminal-pair reliability algorithms fall into two categories: *Path-based*, such as those in [16], [17], and *cut-based*, such as those in [18], [19]. In path- or cut-based methods, in the first step of the computation of the terminal-pair reliability, all minimal paths or cutsets between source and destination machines are enumerated. Then, either an exact or an approximate reliability expression based on minimal paths or cutsets is derived and evaluated accordingly. However, these algorithms are computationally expensive to include in a matching and scheduling algorithm. In this section, a computationally inexpensive method to estimate the reliability of communication between two machines is presented.

Let $p_{s,t}^i = \{r_1^i, r_2^i, \dots, r_{l_i}^i\}$, $1 \leq i \leq \gamma$, denote the i th simple path between machines m_s and m_t , where γ denotes the number of different simple paths between these two machines and $r_1^i = m_s$ and $r_{l_i}^i = m_t$, $1 \leq i \leq \gamma$. Let $E_{s,t}$ denote the event that the communication between m_s and m_t is failure-free and $E_{s,t}^i$ denote the event that the i th simple path between m_s and m_t is failure-free. The probability that the communication between m_s and m_t is failure-free, which is denoted by $P[E_{s,t}]$, is the probability that at least one of the simple paths is failure-free and, using the *inclusion-exclusion* principle, $P[E_{s,t}]$ can be computed as:

$$\begin{aligned} P[E_{s,t}] &= P[E_{s,t}^1 \cup \dots \cup E_{s,t}^\gamma] \\ &= \sum_{i=1}^{\gamma} P[E_{s,t}^i] - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} P[E_{s,t}^i \cap E_{s,t}^j] \\ &\quad + \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} P[E_{s,t}^i \cap E_{s,t}^j \cap E_{s,t}^k] - \dots \\ &\quad + (-1)^{\gamma-1} P[E_{s,t}^1 \cap \dots \cap E_{s,t}^\gamma]. \end{aligned} \quad (4)$$

It is important to note that, although resource failures are statistically independent, the failure of simple paths may not be statistically independent. This is due to the fact that a resource can appear in more than one path. Let E_{r_i} denote the event that resource r_i is failure-free. From set theory, recall that $E_{r_i} \cap E_{r_i} = E_{r_i}$. Assuming that the failures of resources are statistically independent,

$$P[E_{s,t}^1 \cap \dots \cap E_{s,t}^l] = \prod_{r_i \in \mathcal{R}(p_{s,t})} P[E_{r_i}], \quad l \leq \gamma, \quad (5)$$

where $\mathcal{R}(p_{s,t}) = p_{s,t}^1 \cup \dots \cup p_{s,t}^l$ denotes the set of resources that makes up paths $p_{s,t}^1, \dots, p_{s,t}^l$.

Each term in (4) can be computed as follows: Let τ_{r_i} be a random variable which denotes the time of failure of resource r_i , $f_{\tau_{r_i}}(t)$ denote the probability density function of random variable τ_{r_i} , and $R_{\tau_{r_i}}(t)$ denote the reliability of resource r_i , i.e., the probability that resource r_i is operational for a length of time t . In a Poisson random process, random variable τ_{r_i} has an exponential probability density

function $f_{\tau_{r_i}}(t) = \lambda_{r_i} e^{-\lambda_{r_i} t}$. Consequently, the reliability of a resource is given by $R_{\tau_{r_i}}(t) = e^{-\lambda_{r_i} t}$ [23].

Let $\tau_{s,t}^i$ be a random variable which denotes the time of failure of the i th simple path between m_s and m_t , and $R_{\tau_{s,t}^i}(t)$ denote the reliability of the i th simple path between m_s and m_t . Since the failure of resources are statistically independent, the reliability of a simple path is the product of the reliability of the resources which make up that simple path.

$$\begin{aligned} R_{\tau_{s,t}^i}(t) &= P[E_{s,t}^i] = \prod_{j=1}^{l_i} R_{\tau_{r_j^i}}(t) = e^{-\left(\sum_{j=1}^{l_i} \lambda_{r_j^i}\right)t} \\ &= e^{-\left(\sum \lambda_{s,t}^i\right)t}, \end{aligned} \quad (6)$$

where $\lambda_{s,t}^i$ denotes the set of failure rates of the resources which makes up the path, i.e., $\lambda_{s,t}^i = \{\lambda_{r_1^i}, \dots, \lambda_{r_{l_i}^i}\}$, and $\sum\{A\}$ denotes the summation of the elements of set A . Using (5) and (6), the reliability term corresponding to the intersection of events can be computed by

$$\begin{aligned} P[E_{s,t}^1 \cap \dots \cap E_{s,t}^l] &= \prod_{r_i \in \mathcal{R}(p_{s,t})} P[E_{r_i}] = e^{-\left(\lambda_{r_1} + \dots + \lambda_{r_{|\mathcal{R}(p_{s,t})|}}\right)t} \\ &= e^{-\left(\sum \Lambda_{s,t}\right)t} \end{aligned} \quad (7)$$

where $\Lambda_{s,t} = \lambda_{s,t}^1 \cup \dots \cup \lambda_{s,t}^l$ denotes the set of failure rates corresponding to the resources in set $\mathcal{R}(p_{s,t})$, and $|\mathcal{R}(p_{s,t})|$ denotes the number of elements in set $\mathcal{R}(p_{s,t})$. Let $\tau_{s,t}$ be a random variable which denotes the time of failure of the communication between m_s and m_t , and $R_{\tau_{s,t}}(t)$ denote the reliability of the communication between m_s and m_t . Finally, substituting (6) and (7) for the first term and the other terms, respectively, (4) can be written as:

$$\begin{aligned} R_{\tau_{s,t}}(t) &= P[E_{s,t}] = \sum_{i=1}^{\gamma} e^{-\left(\sum \lambda_{s,t}^i\right)t} - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} e^{-\left(\sum \lambda_{s,t}^i \cup \lambda_{s,t}^j\right)t} \\ &\quad + \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} e^{-\left(\sum \lambda_{s,t}^i \cup \lambda_{s,t}^j \cup \lambda_{s,t}^k\right)t} - \dots \\ &\quad + (-1)^{\gamma-1} e^{-\left(\sum \lambda_{s,t}^1 \cup \dots \cup \lambda_{s,t}^\gamma\right)t}. \end{aligned} \quad (8)$$

However, computing an exact reliability expression given by (8) for an arbitrary network is *NP-hard*. To simplify the computation of $R_{\tau_{s,t}}$, e^x can be replaced by its small-value approximation, $1+x$, and the reliability expression becomes

$$R_{\tau_{s,t}}(t) = 1 - \left[\sum \left\{ \bigcap_{i=1}^{\gamma} \lambda_{s,t}^i \right\} \right] t. \quad (9)$$

The detailed derivation of (9) is given in Appendix A. For small values of x , the small-value approximation is quite accurate. For larger values ($|x| > 1$), the approximation will underestimate the exact reliability of the communication. For an arbitrary large t , $R_{\tau_{s,t}}(t) < 0$ is possible, in which case $R_{\tau_{s,t}}(t) = 0$ is assumed.

If both m_s and m_t are in the same biconnected component of the network, all simple paths between m_s and m_t have only m_s and m_t as the common resources.

Therefore, $\sum\{\bigcap_{i=1}^{\gamma} \lambda_{s,t}^i\} = \lambda_{m_s} + \lambda_{m_t}$. For example, there are three different simple paths between machines 7 and 10 in Fig. 1, that is, $p_{7,10}^1 = \{m_7, n_{7,9}, m_9, n_{9,10}, m_{10}\}$, $p_{7,10}^2 = \{m_7, n_{7,8}, m_8, n_{8,11}, m_{11}, n_{11,10}, m_{10}\}$, and $p_{7,10}^3 = \{m_7, n_{7,8}, m_8, n_{8,11}, m_{11}, n_{11,9}, m_9, n_{9,10}, m_{10}\}$. Note that $\bigcap_{i=1}^3 p_{7,10}^i = \{m_7, m_{10}\}$. As a result, the complexity of computing (9) is $O(1)$.

If m_s and m_t are in different biconnected components of the network, all simple paths between m_s and m_t have the following resources as the common resources; a set of machines, each of which corresponds to an articulation point connecting two biconnected components, and a set of links, each of which corresponds to a bridge, along the shortest path between m_s and m_t . For example, there are 48 different simple paths between machines 2 and 15 in Fig. 1 and $\bigcap_{i=1}^{48} p_{2,15}^i = \{m_2, m_6, n_{6,7}, m_7, m_{10}, n_{10,13}, m_{13}, m_{15}\}$. Therefore, in order to compute the reliability of the communication between m_2 and m_{15} , all those articulation points and bridges along the shortest path must be determined. Consequently, complexity of computing (9) is $O(|M| + |N|)$.

6 COMPUTATION OF $C(v_i, m_j)$ TERM

This section presents cost functions which can guide a matching and scheduling algorithm to produce schedules such that failures of the network resources will have less effect on the execution of the application.

6.1 Cost Functions

Let $H = \{h_1, \dots, h_f\}$, $f \leq n$, denote a set of matching and scheduling decisions and $C_k(h_i)$ denote the cost of a matching and scheduling decision $h_i \in H$ with respect to the k th cost function. Each element $h_i \in H$ is a tuple of the form $(v_i, m_j, \mathcal{R}(v_i), s_i, f_i)$, where m_j represents the machine to which task v_i is assigned and $\mathcal{R}(v_i) = \bigcap_{k=1}^{\gamma} p_{j,src}^k$ represents a subset of resources that are identified by Definition 4.

The first cost function is due to Theorem 1. Since the reliability of the communication between machines m_j on which a task is executing and m_{src} is crucial for the reliability of the application, the first cost function is defined to be the time that the application will lose due to the failure of network resources between m_j and m_{src} . That is, the cost will be equal to the expected time of failure of the communication between machines m_j and m_{src} , where the time of failure of the communication $\tau_{j,src}$ is defined relative to s_i . As a result, the cost of a matching and scheduling decision with respect to the first cost function, $C_1(h_i)$, $h_i = (v_i, m_j, \mathcal{R}(v_i), s_i, f_i)$, is defined to be:

$$\begin{aligned} C_1(h_i) &\triangleq \int_{s_i}^{f_i} t f_{\tau_{j,src}}(t - s_i) dt \\ &= -t R_{\tau_{j,src}}(t - s_i) \Big|_{s_i}^{f_i} + \int_{s_i}^{f_i} R_{\tau_{j,src}}(t - s_i) dt. \end{aligned} \quad (10)$$

For an arbitrary network, the computation of $C_1(h_i)$ given by (10) is at least as hard as the computation of the reliability of the communication between machines m_j and m_{src} . To simplify the computation of $C_1(h_i)$, e^x is replaced by its small-value approximation and the cost function becomes

$$C_1(h_i) = \left[\sum \left\{ \prod_{k=1}^{\gamma} \{\lambda_{j,src}^k\} \right\} \right] (f_i - s_i) f_i. \quad (11)$$

The second cost function is due to the following heuristics: 1) Each task, in general, will have a different effect on the reliability of the application. For example, a task which has more successor tasks will usually depend on more resources for communicating with its successor tasks. Definition 4, however, takes only the reliability of resources between machines m_j , on which a task is executing, and m_{src} into account. If a task which is important for the reliability of the application is being scheduled, this task should be scheduled on more reliable resources. In such a case, the cost should increase. 2) If the communication between a machine on which a task is scheduled and the source machine m_{src} is highly reliable, the cost should decrease. 3) If the duration of the execution of task is long, the cost should increase. Thus, the second cost function $C_2(h_i)$, $h_i = (v_i, m_j, \mathcal{R}(v_i), s_i, f_i)$, is defined to be:

$$C_2(h_i) \triangleq w_R(v_i) (1 - R_{\tau_{j,src}}(f_i - s_i)) (f_i - s_i), \quad (12)$$

where $w_R(v_i) > 0$ is the weight of reliability of task v_i , which is introduced to meet the first heuristic. That is, $w_R(v_i)$ represents the effect of task v_i on the overall reliability of the application. Note that $\mathcal{R}(v_i)$ is the same as the one that is defined for the first cost function.

The cost function defined by (11) does not consider the information in the task graph explicitly while computing the cost of a matching and scheduling decision. This is addressed to some degree in designing the second cost function by introducing the term $w_R(v_i)$. Motivated by this fact, the third cost function presented in the following is designed to use the information in the task graph explicitly.

Let $IP(v_i)$ denote a set which includes all immediate predecessor tasks of task v_i and $f_{k,i}$ denote the time when task $v_k \in IP(v_i)$ has finished transmitting all of its results to task v_i . $f_{k,i}$ is defined to be:

$$f_{k,i} = \max\{s_i, t_k^F\} + t_{k,i}^N, \quad (13)$$

where t_k^F denotes the time when the execution of task v_k has finished and $t_{k,i}^N$ denotes the amount of time needed to transfer all relevant data from machine m_p , on which task v_k is executing, to m_j . Definition 4 is modified accordingly as follows:

Definition 7. A task $v_i \in V$ executing on machine m_j is defined to be failure-free if task $v_k \in IP(v_i)$ is able to find a failure-free path to task v_i during time interval $[s_i, f_{k,i}]$ and machine m_j is failure-free during time interval $[s_i, f_i]$.

Definition 7 is an intuitive one in that task v_i is assumed to be failure-free if 1) it successfully receives data from its predecessor tasks and 2) the machine to which task v_i is assigned is failure-free during the execution of task v_i and the communication between task v_i and its successor tasks. It is important to note that Theorem 1 holds for Definition 7 as well. Since the proof is trivial, it is omitted.

According to Definition 7, the reliability of the machine (m_j) on which task $v_i \in V$ is executing and the communication between tasks $v_k \in IP(v_i)$ and v_i are crucial for the reliability of the application. Thus, the third cost function is defined to be the time that the application will lose due to

the failure of machine m_j or the failure of network resources used during the intertask communication between tasks $v_k \in IP(v_i)$ and v_i . As a result, the cost of a matching and scheduling decision with respect to the third cost function, $C_3(h_l)$, $h_l = (v_i, m_j, \mathcal{R}(v_i), s_i, f_i)$, is defined to be:

$$\begin{aligned} C_3(h_l) &\triangleq \max \left\{ \int_{s_i}^{f_i} t f_{\tau_{m_j}}(t - s_i) dt, \max_{v_k \in IP(v_i)} \right. \\ &\quad \left. \left\{ \int_{s_i}^{f_{k,i}} t f_{\tau_{p,j}}(t - s_i) dt \right\} \right\} \\ &= \max \left\{ \lambda_{m_j} (f_i - s_i) f_i, \max_{v_k \in IP(v_i)} \right. \\ &\quad \left. \left\{ \left[\sum_{i=1}^{\gamma_{p,j}} \left\{ \bigcap_{j=1}^{\gamma_{p,j}^i} \lambda_{p,j}^i \right\} \right] (f_{k,i} - s_i) f_{k,i} \right\} \right\}, \end{aligned} \quad (14)$$

where $\gamma_{p,j}$ denotes the number of simple paths between machines m_p and m_j . In (14), the first term is the expected cost of failure of machine m_j and the second term is the expected cost of failure of communication between machines m_p and m_j . For the third cost function, $\mathcal{R}(v_i)$ is defined to be:

$$\mathcal{R}(v_i) = \begin{cases} m_j, & \text{if } \lambda_{m_j} (f_i - s_i) f_i \geq \max_{v_k \in IP(v_i)} \\ & \left\{ \left[\sum_{i=1}^{\gamma_{p,j}} \left\{ \bigcap_{j=1}^{\gamma_{p,j}^i} \lambda_{p,j}^i \right\} \right] (f_{k,i} - s_i) f_{k,i} \right\} \\ \bigcap_{i=1}^{\gamma_{p^*,j}} p_{p^*,j}^i, & \text{otherwise,} \end{cases} \quad (15)$$

where $v_{k^*} - m_{p^*}$ denote a task-machine pair which maximizes the second term in (14).

6.2 Incremental Cost Function

The cost functions defined by (11), (12), and (14) can be used to associate a cost with a single matching and scheduling decision. However, a solution to the matching and scheduling problem consists of more than one matching and scheduling decision. To model the contribution of the different matching and scheduling decisions to the unreliability of the application, an incremental cost function, which is denoted by $C_{\Delta_k}(H, h_l)$, is devised. If a new $h_l = (v_i, m_j, \mathcal{R}(v_i), s_i, f_i)$ is added to set H , the incremental cost function $C_{\Delta_k}(H, h_l)$ is defined to be:

$$C_{\Delta_k}(H, h_l) = C_k(h_l) - \sum_{r_x \in \mathcal{R}(v_i)} C_k(v_i, r_x, s_i, f_i), \quad k = 1, 2, 3, \quad (16)$$

where $C_k(v_i, r_x, s_i, f_i)$ denotes the overstated cost with respect to the k th cost function of using resource r_x in the time interval $[s_i, f_i]$ because of task v_i . The computation of the $C_k(v_i, r_x, s_i, f_i)$ term is discussed in Appendix B in detail. The rationale behind the $C_k(v_i, r_x, s_i, f_i)$ term is as follows: It is likely that there can be more than one $h_l \in H$ for a single resource which have overlapping time intervals. In order to compute the incremental cost precisely, extra costs due to the overlapping time intervals for the resources in set $\mathcal{R}(v_i)$ must be subtracted from $C_k(h_l)$.

Because of the way that the $C(v_i, m_j)$ term is introduced in Section 3.2, let $C(v_i, m_j) = C_{\Delta_k}(H, h_l)$. Note that since the $C(v_i, m_j)$ term defines the effect of a scheduling decision on the reliability of the application in *time units*, the scheduling algorithms presented in this study are significantly different

from the previous ones. Furthermore, the $C(v_i, m_j)$ term is not exclusively formulated for the DLS algorithm in that it can be incorporated into other static or dynamic matching and scheduling algorithms provided that they deploy a cost function in terms of time units. Finally, incorporating $C(v_i, m_j)$ into a particular matching and scheduling algorithm enables that algorithm to consider the effect of different scheduling decisions on the reliability of the application.

7 SIMULATIONS

In order to evaluate the proposed algorithms, a simulation program that can be used to emulate the execution of randomly created 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 machines p_M and the number of switches p_S ($p_M + p_S = p$). The variable p_M determines the number of heterogeneous machines on which a task can be executed. Associated with each machine is a FIFO queue that holds the tasks that are scheduled on this machine. In addition, each machine is assumed to execute a task in its queue to completion without preemption. In order to interconnect p_M machines, a network of p_S switches is employed, where the network topology is randomly generated and each machine is randomly connected to a switch. This simulation model closely mimics a computing system where a set of machines is interconnected by a switched-based network. Note that the network itself can also be heterogeneous, where several networking technologies are simultaneously used. 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 1×10^{-3} and 1×10^{-4} failures/hr [20]. In addition, the transmission rates of links are assumed to be uniformly distributed between 1 and 10 Mbits/sec.

The simulation studies performed are grouped into three sets; 1) executing randomly generated task graphs with a different number of tasks (between 10 and 50) on a computing system with $p_M = 10$ and $p_S = 20$, 2) executing randomly generated task graphs with 30 tasks on a computing system with p_M ranging from 10 to 50 and $p_S = 20$, and 3) executing a real application task graph on a computing system with p_M ranging from 10 to 50 and $p_S = 20$. With respect to the first set of simulations, the number of tasks in a randomly generated task graph is fixed to 10, 20, 30, 40, or 50 and a source machine is randomly associated with the task graph. The execution time of each task of the task graph is assumed to be uniformly distributed between 10 and 120 minutes, where the execution times of a given task are different on different machines. The volume of data to be transmitted among tasks are randomly generated such that the *communication to computation ratio* (CCR) is 0.1, 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 times CCR. By using a range of CCR values, different types of applications [24] can be accommodated. That is, computation intensive applications may be modeled by assuming CCR = 0.1, whereas, data intensive applications may be modeled by assuming CCR = 10.0. In the simulations, w_R ,

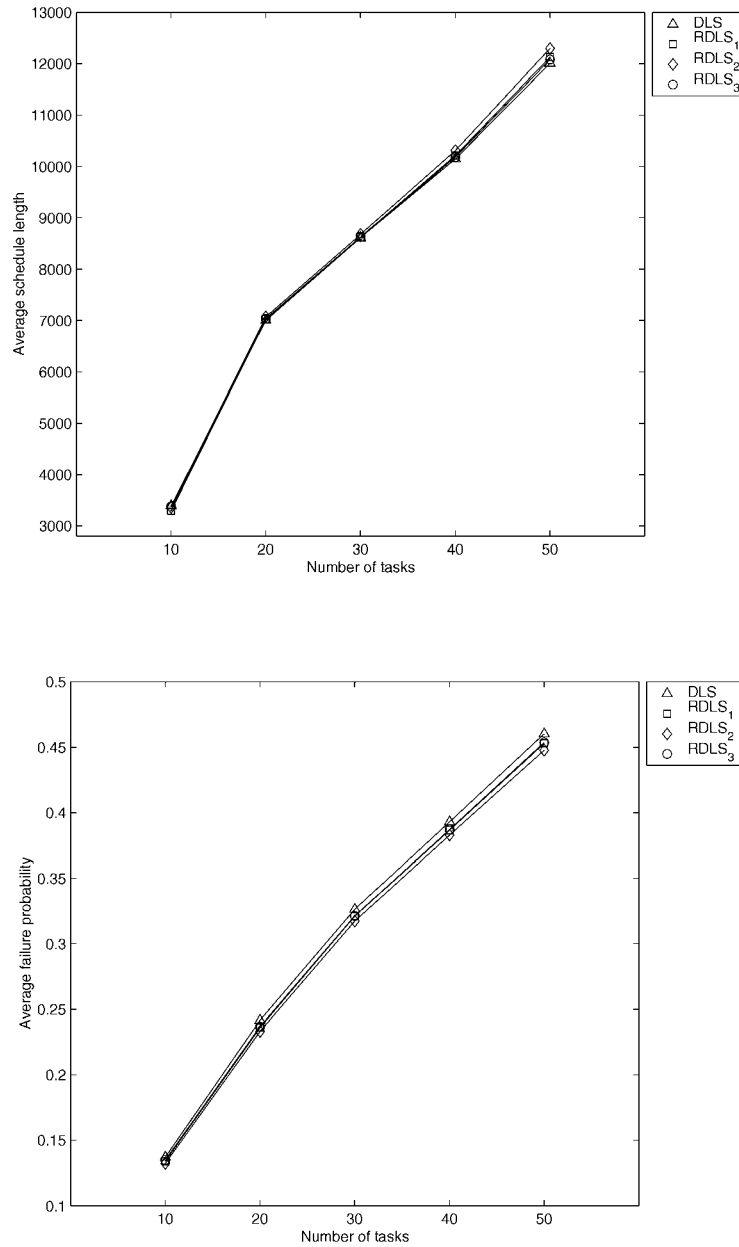


Fig. 2. Average schedule length and failure probability of DLS and RDLS algorithms for CCR = 0.1.

the weight of reliability of a task used in the second cost function (12) is computed as follows: The median execution time of task v_i across all machines and the estimated time to transmit all relevant data from task v_i to its successors are added. This sum is normalized by the maximum of the sums obtained across all tasks and the normalized value plus one is assigned as $w_R(v_i)$.

For the first set of simulation studies, the results are shown in Figs. 2, 3, and 4, where each data point is the average of the data obtained in 1,000 experiments. Note that RDLS_k, $1 \leq k \leq 3$, in Figs. 2, 3, 4, 5, 7, and 8 denotes the RDLS algorithm using the k th cost function. According to the simulation results, the performance of proposed algorithms varies with respect to the application size and the CCR. For CCR = 0.1, the performance of DLS and RDLS algorithms is identical. This is due to the fact that relatively small durations of execution of tasks make

the incremental cost function small as compared to other terms in the dynamic level expression given by (3). Consequently, the dynamic level expression is dominated by the first three terms and the RDLS algorithm reduces to the DLS algorithm. For CCR = 1.0 and CCR = 10.0, however, the impact of the $C(v_i, m_j)$ term on scheduling decisions is clear, that is, the incorporation of $C(v_i, m_j)$ results in significant drops in the failure probability of applications at the expense of increasing schedule lengths. Specifically, Table 1 shows the percentage of increase in the average schedule length and the percentage of decrease in the failure probability of applications under RDLS as compared to DLS. According to Table 1, for CCR = 1.0, the failure probability of applications decreases nearly at the same rate as the increase in the execution time except for the second cost function. This experiment clearly reveals the trade-off between the execution time and failure probability

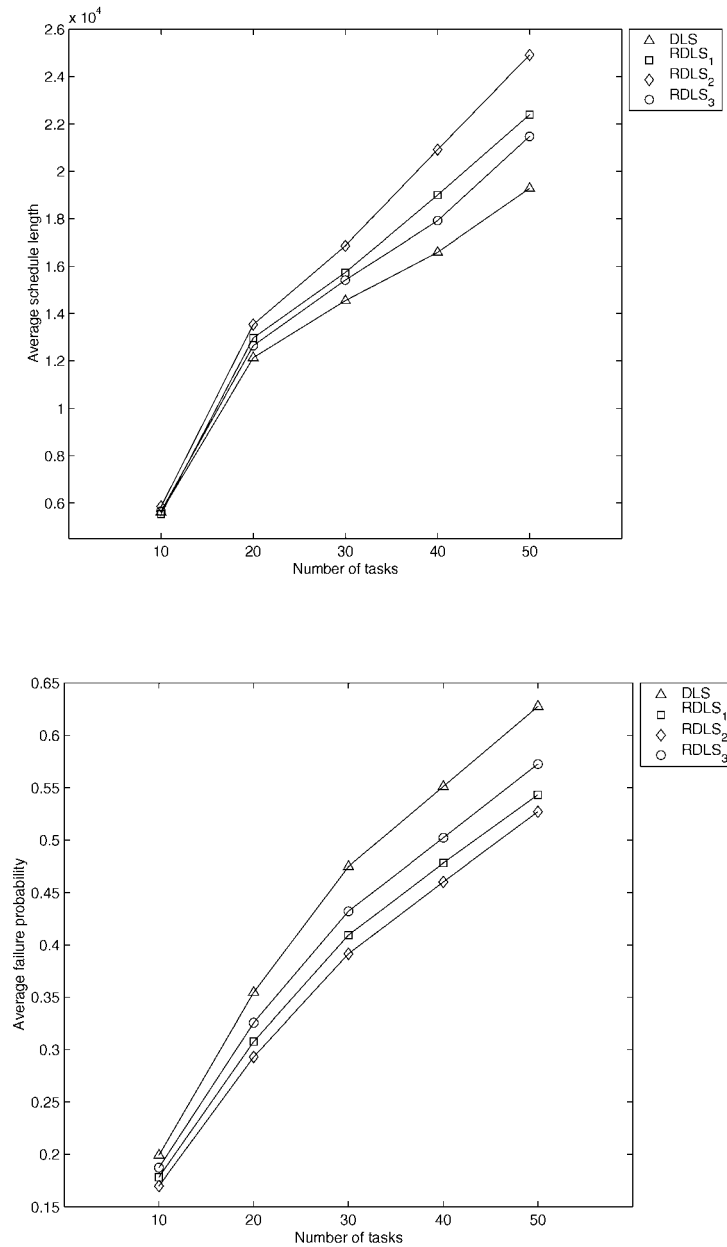


Fig. 3. Average schedule length and failure probability of DLS and RDLS algorithms for $CCR = 1.0$.

of applications. For $CCR = 10.0$, RDLS is extremely efficient, where the amount of the increase in the schedule length is about half of the amount of the decrease in the failure probability.

For the second set of simulation studies, random task graphs with 30 tasks and $CCR = 1.0$ are assumed to be executed on a heterogeneous computing system with 10, 20, 30, 40, or 50 machines. The results of this simulation study are shown in Fig. 5. According to Fig. 5, while the number of machines increases, the average schedule length of DLS and RDLS decreases as expected. In parallel to this decrease, the failure probability of applications also decreases. Specifically, Table 2 shows the performance comparison of the DLS and RDLS algorithms.

For the third set of simulation studies, CSTEM (Coupled Structural-Thermal-Electromagnetic Analysis and Tailoring

of Graded Composite Structures) [25] application, which is a finite element-based computer program, is used. CSTEM analyzes and optimizes the performance of composite structures using a variety of dissimilar analysis modules, including a structural analysis module, a thermal analysis module, an electromagnetic absorption analysis module, and an acoustic analysis module. The task graph of CSTEM is shown in Fig. 6, where the numbers within the nodes of the graph represent the approximate execution time, in seconds, on a Sun Microsystems Sparc 10 workstation [25]. Note that the execution time of CSTEM depends on the size of the problem. During the simulations, CSTEM's task graph with scaled up execution times is used. The execution times of tasks are determined as follows: The task with the smallest execution time (0.3 seconds) is assumed to take 10 minutes on the average for a given

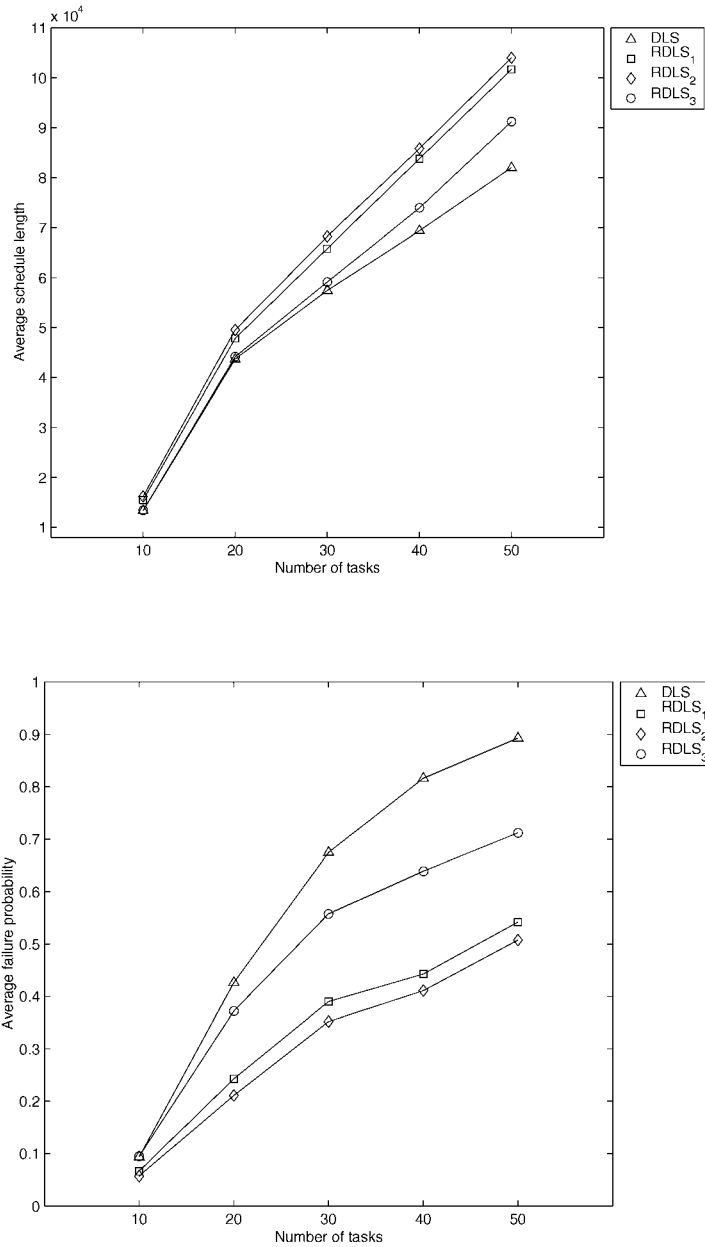


Fig. 4. Average schedule length and failure probability of DLS and RDLS algorithms for CCR = 10.0.

problem size. Then, the average execution times (in seconds) of the other tasks are found by multiplying their execution times given in the task graph by $\frac{10 \times 60}{0.3}$. Under these settings, the simulation results are presented in Figs. 7 and 8 for CCR = 0.1 and CCR = 1.0, respectively. According to the simulation results, similar to the previous experiments, while the failure probability of CSTEM is decreasing, its schedule length is increasing. In addition, the decrease in the schedule length of CSTEM with increasing number of machines usually makes CSTEM's failure probability decrease. Specifically, a performance comparison of the DLS and RDLS algorithms is depicted in Table 3.

In general, the results of simulation studies can be summarized as follows:

1. The performance of the RDLS algorithm heavily depends on the CCR. For small values of the CCR, RDLS performs similar to DLS, whereas, for large values of the CCR, the RDLS algorithm is preferable due to the fact that it considerably reduces the failure probability at the expense of relatively small increase in the execution time of applications.
2. The performance difference between the RDLS algorithm and DLS algorithm increases in parallel to the increase in the size of the application.
3. There is a trade-off between the execution time and failure probability of applications. As a result, in general, both objectives cannot be minimized at the same time.
4. The RDLS₂ algorithm outperforms both RDLS₁ and RDLS₃ algorithms in terms of minimizing the failure probability at the price of longer execution times.

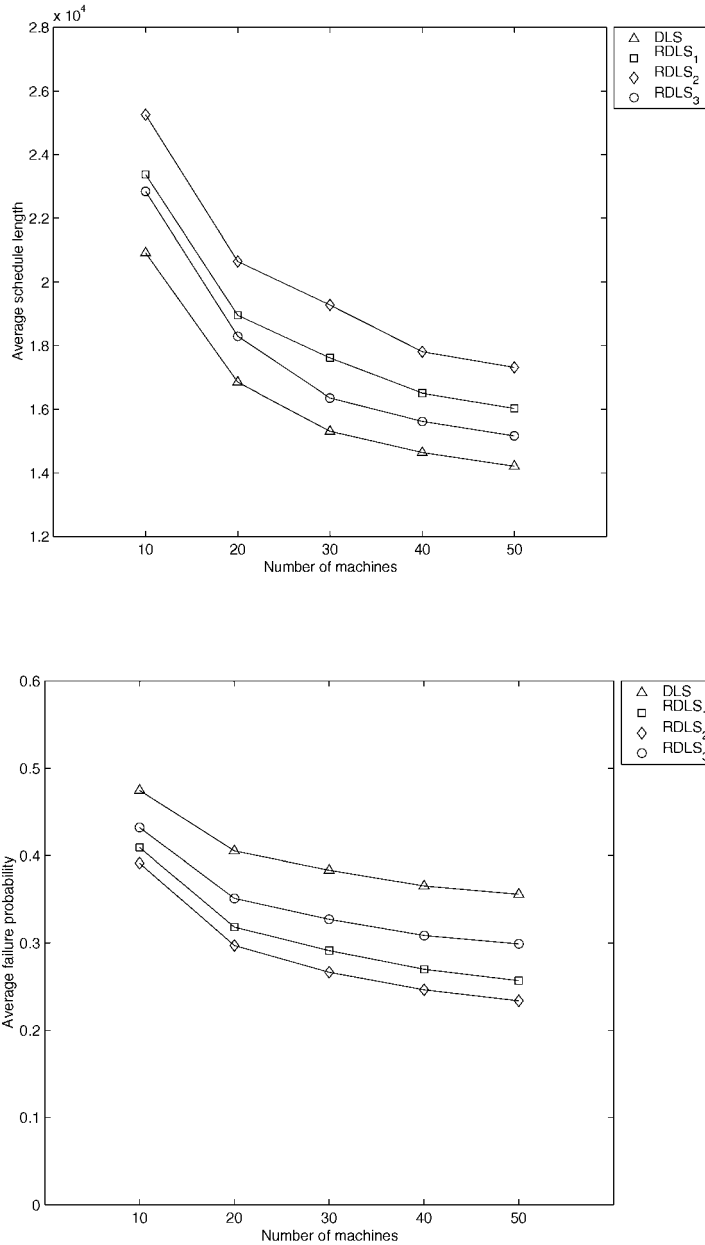


Fig. 5. Average schedule length and failure probability of DLS and RDLS algorithms with respect to p_M .

The order between RDLS₁ and RDLS₃ algorithms depends on the structure of the application.

8 CONCLUSIONS

From the simulation results presented in the previous section, it is clear that the proposed incremental cost function can be used to decrease the failure probability of the task assignments produced by existing matching and scheduling algorithms. The unique features of the incremental cost function presented in this study are 1) it is not restricted to tree-based networks, 2) it can be used with any matching and scheduling algorithm which expresses its cost function in terms of time units, and 3) it can be used to find compromise solutions to the problem of minimizing the execution time and probability of failure of an application. Furthermore, a computationally simple method has also

been developed to determine the reliability of the communication between two machines in an arbitrary network.

The main contribution of this study to scheduling systems is that it extends the traditional formulation of the scheduling problem so that both execution time and reliability of applications are simultaneously accounted for. The simulations show that for applications with relatively small execution times, the effect of the incremental cost function on scheduling decisions is usually small, which results in making the minimization of the execution time the primary objective. On the other hand, for large applications with long execution times, it is likely that a user may demand a compromise task assignment in terms of minimizing both execution time and failure probability. It is shown in the previous section that the proposed algorithms are capable of producing such task assignments.

The current trend in designing scheduling algorithms is to respect users' demands, that is, provide Quality of

TABLE 1
Average Performance Comparison of RDLS with Respect to DLS in Terms of Application Execution Time and Failure Probability for CCR = 1.0 and CCR = 10.0

CCR	Cost function	Increase in schedule length	Decrease in failure probability
1.0	1	16%	14%
	2	29%	18%
	3	11%	9%
10.0	1	24%	46%
	2	27%	51%
	3	11%	22%

Service (QoS)-based scheduling and this study can be considered to be a step in that direction. In implementing a scheduler, however, other challenging problems, including task profiling for a given application, analytical benchmarking of the machines in the system, machine and system load estimation, resource management, etc., need to be addressed. Although individual solutions to each of these problems have been proposed [26], [27], [28], growing size of heterogeneous computing makes it difficult to find efficient solutions to these problems that can be implemented in a practical system. How each of these problems is solved will have an impact on the performance of the scheduling algorithm deployed in a heterogeneous system. In particular, the predictability of the performance of an application under a scheduling algorithm depends on the quality of information that is provided to the scheduler about the application and the heterogeneous system. As a result, there are still challenges that need to be overcome to implement an efficient scheduler for a large heterogeneous computing system.

APPENDIX A

SIMPLIFYING THE EXACT RELIABILITY EXPRESSION USING THE SMALL-VALUE APPROXIMATION

When $1 + x$ is substituted for e^x , (8) becomes

$$\begin{aligned}
 R_{\tau_{s,t}}(t) &= \sum_{i=1}^{\gamma} \left(1 - \sum \{\lambda_{s,t}^i\} t\right) - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} \left(1 - \sum \{\lambda_{s,t}^i \cup \lambda_{s,t}^j\} t\right) \\
 &+ \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} \left(1 - \sum \{\lambda_{s,t}^i \cup \lambda_{s,t}^j \cup \lambda_{s,t}^k\} t\right) - \dots \\
 &+ (-1)^{\gamma-1} \left(1 - \sum \{\lambda_{s,t}^1 \cup \dots \cup \lambda_{s,t}^{\gamma}\} t\right).
 \end{aligned} \tag{17}$$

TABLE 2

Average Performance Comparison of RDLS with Respect to DLS in Terms of Application Execution Time and Failure Probability for the Second Set of Simulations

Cost function	Increase in schedule length	Decrease in failure probability
1	15%	28%
2	26%	34%
3	9%	16%

Using $\binom{\gamma}{1} - \binom{\gamma}{2} + \dots + (-1)^{\gamma-1} \binom{\gamma}{\gamma} = 1$ and the inclusion-exclusion principle, (17) can be written as:

$$\begin{aligned}
 R_{\tau_{s,t}}(t) &= 1 - \sum \left\{ \sum_{i=1}^{\gamma} \lambda_{s,t}^i - \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} [\lambda_{s,t}^i + \lambda_{s,t}^j - (\lambda_{s,t}^i \cap \lambda_{s,t}^j)] \right. \\
 &+ \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} [\lambda_{s,t}^i + \lambda_{s,t}^j + \lambda_{s,t}^k - (\lambda_{s,t}^i \cap \lambda_{s,t}^j) - (\lambda_{s,t}^i \cap \lambda_{s,t}^k) \\
 &- (\lambda_{s,t}^j \cap \lambda_{s,t}^k) + (\lambda_{s,t}^i \cap \lambda_{s,t}^j \cap \lambda_{s,t}^k)] - \dots + (-1)^{\gamma-1} \cdot \left[\sum_{i=1}^{\gamma} \lambda_{s,t}^i \right. \\
 &- \sum_{i=2}^{\gamma} \sum_{j=1}^{i-1} (\lambda_{s,t}^i \cap \lambda_{s,t}^j) + \sum_{i=3}^{\gamma} \sum_{j=2}^{i-1} \sum_{k=1}^{j-1} (\lambda_{s,t}^i \cap \lambda_{s,t}^j \cap \lambda_{s,t}^k) - \dots \\
 &\left. \left. + (-1)^{\gamma-1} (\lambda_{s,t}^1 \cap \dots \cap \lambda_{s,t}^{\gamma}) \right] \right\} t.
 \end{aligned} \tag{18}$$

The number of $\lambda_{s,t}^i$, $1 \leq i \leq \gamma$, is

$$1 - \binom{\gamma-1}{1} + \binom{\gamma-1}{2} - \dots - (-1)^{\gamma-1} \binom{\gamma-1}{\gamma-1} = 0;$$

the number of $\lambda_{s,t}^i \cap \lambda_{s,t}^j$, $1 \leq i, j \leq \gamma$ and $i \neq j$, is

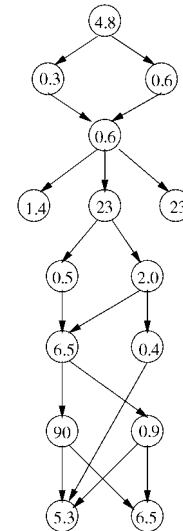


Fig. 6. Task graph of CSTEM application.

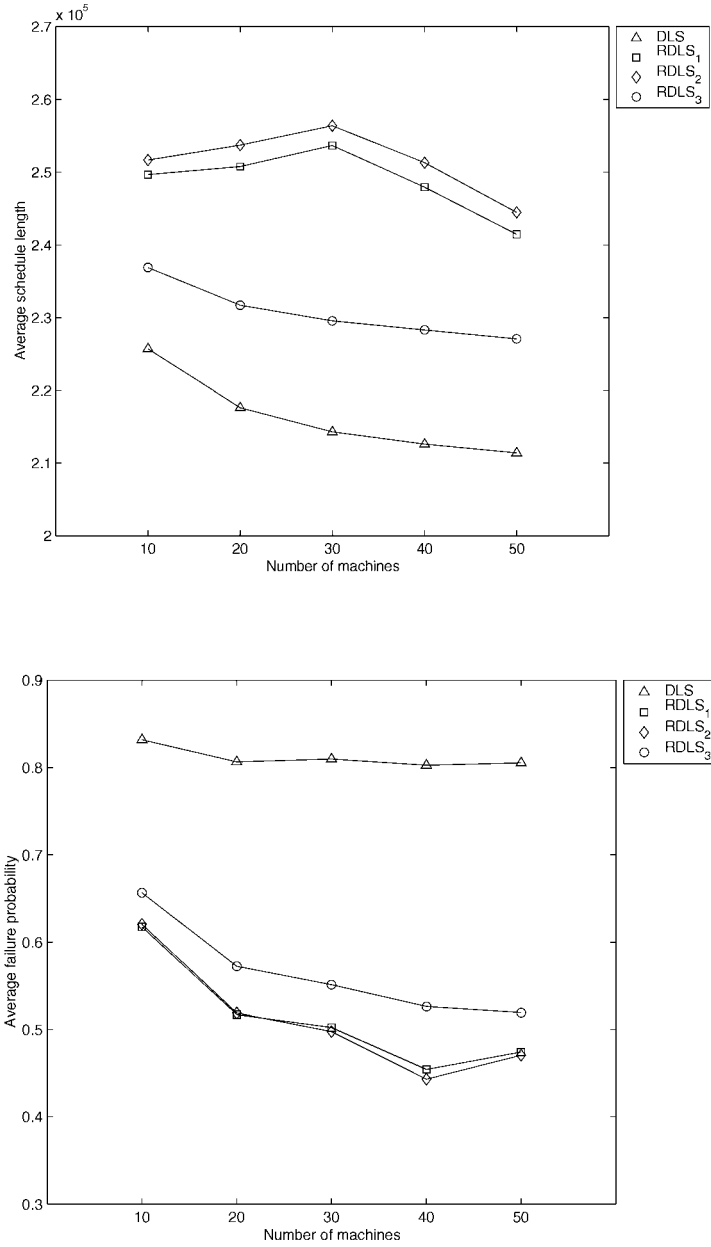


Fig. 7. Average schedule length and failure probability of DLS and RDLS algorithms for CSTEM application with CCR = 0.1.

$$-1 + \binom{\gamma-2}{1} - \binom{\gamma-2}{2} + \dots + (-1)^{\gamma-1} \binom{\gamma-2}{\gamma-2} = 0;$$

the number of $\lambda_{s,t}^i \cap \lambda_{s,t}^j \cap \lambda_{s,t}^k$, $1 \leq i, j, k \leq \gamma$ and $i \neq j \neq k$, is

$$1 - \binom{\gamma-3}{1} + \binom{\gamma-3}{2} - \dots - (-1)^{\gamma-1} \binom{\gamma-3}{\gamma-3} = 0;$$

and so on. Eventually, all terms but one will cancel out. The only term which will remain is

$$(-1)^{\gamma-1} \times (-1)^{\gamma-1} (\lambda_{s,t}^1 \cap \dots \cap \lambda_{s,t}^\gamma).$$

Thus, (8) simplifies to (9).

APPENDIX B

COMPUTING THE OVERSTATED COST

Let \hat{s}_i and \hat{f}_i denote the start and finish times when the resource r_x must remain failure-free before a new h_i is added to set H . The cost of using resource r_x with respect to the k th cost function in time interval $[\hat{s}_i, \hat{f}_i]$, which is denoted by $\hat{c}_k(r_x, \hat{s}_i, \hat{f}_i)$, can be expressed as a summation of the costs of using resource r_x in several time intervals.

$$\hat{c}_k(r_x, \hat{s}_i, \hat{f}_i) = \hat{c}_k(r_x, \hat{t}_1, \hat{t}_2) + \hat{c}_k(r_x, \hat{t}_2, \hat{t}_3) + \dots + \hat{c}_k(r_x, \hat{t}_{\hat{\alpha}-1}, \hat{t}_{\hat{\alpha}}), \quad (19)$$

where $1 \leq k \leq 3$, $\hat{t}_1 = \hat{s}_i$, $\hat{t}_{\hat{\alpha}} = \hat{f}_i$, and $\hat{\alpha} - 1$ denote the number of time intervals in time interval $[\hat{s}_i, \hat{f}_i]$. As an example, suppose that $H = \emptyset$ and

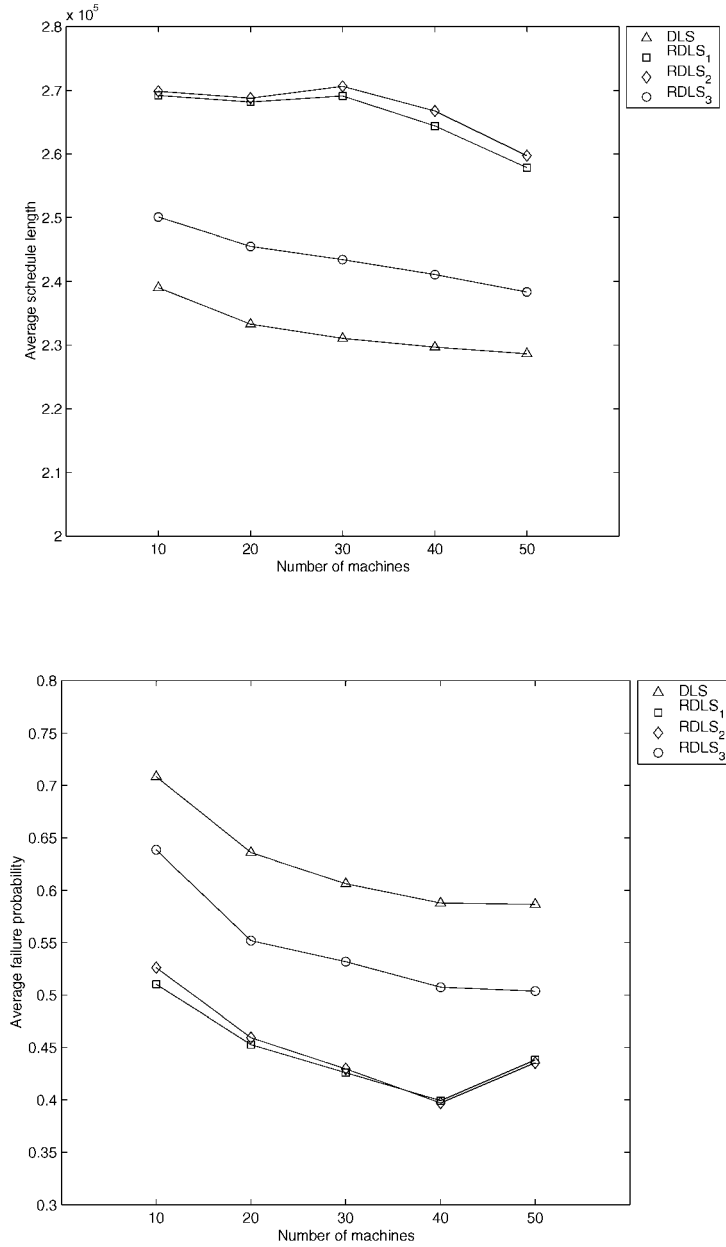


Fig. 8. Average schedule length and failure probability of DLS and RDLS algorithms for CSTEM application with CCR = 1.0.

$$h_1 = (v_1, m_i, \{m_i, m_{src}\}, s_1, f_1)$$

will be added to H . At the beginning, $\hat{s}_i = \hat{f}_i = 0$, $\hat{c}_k(r_x, \hat{s}_i, \hat{f}_i) = 0$ and $\hat{\alpha} = 1, \forall r_x \in \mathcal{R}$. Once h_1 is added,

$$\hat{c}_1(m_i, s_1, f_1) = \lambda_{m_i}(f_1 - s_1)f_1$$

and

$$\hat{c}_1(m_{src}, s_1, f_1) = \lambda_{m_{src}}(f_1 - s_1)f_1.$$

Now, suppose that $h_2 = (v_2, m_j, \{m_j, m_{src}\}, s_2, f_2)$ will be added to $H = \{h_1\}$, where $f_1 > s_2 > s_1$ and $f_2 > f_1$. Once h_2 is added, $\hat{c}_1(m_{src}, s_1, f_2)$ can be written as

$$\begin{aligned} \hat{c}_1(m_{src}, s_1, f_2) &= \hat{c}_1(m_{src}, s_1, s_2) + \hat{c}_1(m_{src}, s_2, f_1) \\ &\quad + \hat{c}_1(m_{src}, f_1, f_2), \end{aligned}$$

where

$$\hat{c}_1(m_{src}, s_1, s_2) = \lambda_{m_{src}}(s_2 - s_1)f_1,$$

$$\hat{c}_1(m_{src}, s_2, f_1) = \lambda_{m_{src}}(f_1 - s_2)f_2, \text{ and}$$

$$\hat{c}_1(m_{src}, f_1, f_2) = \lambda_{m_{src}}(f_2 - f_1)f_2.$$

Similarly, the cost of using resource r_x with respect to the k th cost function in time interval $[s_i, f_i]$, which is denoted by $c_k(r_x, s_i, f_i)$, is expressed as a summation of the costs of using resource r_x in several time intervals.

$$\begin{aligned} c_k(r_x, s_i, f_i) &= c_k(r_x, t_1, t_2) + c_k(r_x, t_2, t_3) \\ &\quad + \dots + c_k(r_x, t_{\alpha-1}, t_\alpha), \end{aligned} \quad (20)$$

where $1 \leq k \leq 3$, $t_1 = s_i$, $t_\alpha = f_i$, and $\alpha - 1$ denote the number of time intervals in time interval $[s_i, f_i]$. Note that

TABLE 3
Average Performance Comparison of RDLS with Respect to
DLS in Terms of Application Execution Time and Failure
Probability for the Third Set of Simulations

CCR	Cost function	Increase in schedule length	Decrease in failure probability
0.1	1	18%	43%
	2	20%	45%
	3	7%	36%
1.0	1	16%	32%
	2	17%	32%
	3	5%	14%

$C_k(h_l) = c_k(r_x, s_i, f_i)$ and, for the third cost function, f_i is defined to be:

$$f_i = \begin{cases} f_i, & \text{if } \lambda_{m_j}(f_i - s_i)f_i \geq \max_{v_k \in IP(v_i)} \\ f_{k^*,i}, & \text{otherwise.} \end{cases} \left\{ \left[\sum_{i=1}^{\gamma_{p,j}} \{ \lambda_{p,j}^i \} \right] (f_{k,i} - s_i) f_{k,i} \right\}$$

In the example given above, while h_2 is added

$$c_1(m_{src}, s_2, f_2) = C_1(h_2) = c_1(m_{src}, s_2, f_1) + c_1(m_{src}, f_1, f_2),$$

where

$$c_1(m_{src}, s_2, f_1) = \lambda_{m_{src}}(f_1 - s_2)f_2$$

and

$$c_1(m_{src}, f_1, f_2) = \lambda_{m_{src}}(f_2 - f_1)f_2.$$

Accordingly, while $C_k(v_i, r_x, s_i, f_i)$ is computed, the overstated cost of using resource r_x must be individually computed in each time interval $[t_j, t_{j+1}]$, $1 \leq j \leq \alpha - 1$. In the example given above, the total overstated cost for h_2 will be

$$\begin{aligned} C(v_2, m_{src}, s_2, f_2) &= C(v_2, m_{src}, s_2, f_1) + C(v_2, m_{src}, f_1, f_2) \\ &= \lambda_{m_{src}}(f_1 - s_2)f_1. \end{aligned}$$

In general, $C_k(v_i, r_x, s_i, f_i)$ is given by

$$C_k(v_i, r_x, s_i, f_i) = \sum_{j=1}^{\alpha-1} C_k(v_i, r_x, t_j, t_{j+1}), \quad k = 1, 2, 3, \quad (21)$$

where,

$$\begin{aligned} &C_k(v_i, r_x, t_j, t_{j+1}) \\ &= \begin{cases} c_k(r_x, t_j, t_{j+1}), & \hat{c}_k(r_x, t_j, t_{j+1}) \geq c_k(r_x, t_j, t_{j+1}) \\ & \text{and } [t_j, t_{j+1}] \in [\hat{s}_i, \hat{f}_i] \\ \hat{c}_k(r_x, t_j, t_{j+1}), & \hat{c}_k(r_x, t_j, t_{j+1}) < c_k(r_x, t_j, t_{j+1}) \\ & \text{and } [t_j, t_{j+1}] \in [\hat{s}_i, \hat{f}_i] \\ 0, & [t_j, t_{j+1}] \notin [\hat{s}_i, \hat{f}_i], \end{cases} \\ &c_k(r_x, t_j, t_{j+1}) = \lambda_{r_x}(t_{j+1} - t_j)f_i, \quad k = 1, 3 \\ &c_2(r_x, t_j, t_{j+1}) = w_R(v_i)\lambda_{r_x}(t_{j+1} - t_j)^2 \frac{(f_i - s_i)^2}{\sum_{j=1}^{\alpha-1} (t_{j+1} - t_j)^2}. \end{aligned} \quad (22)$$

Note that, if $[t_j, t_{j+1}] \notin [\hat{s}_i, \hat{f}_i]$, $\hat{c}_k(r_x, t_j, t_{j+1}) = 0$. Furthermore, it may not be possible to match time interval $[t_j, t_{j+1}]$,

$1 \leq j \leq \alpha - 1$, to time interval $[\hat{t}_u, \hat{t}_{u+1}]$, $1 \leq u \leq \hat{\alpha} - 1$. In such a case, before (21) is evaluated, $\hat{c}_k(r_x, t_j, t_{j+1})$ must be computed as follows:

1. If $s_i > \hat{t}_u$ and $f_i < \hat{t}_{u+1}$, $1 \leq u \leq \hat{\alpha} - 1$,

$$\begin{aligned} \hat{c}_k(r_x, \hat{t}_u, \hat{t}_{u+1}) &= \hat{c}_k(r_x, \hat{t}_u, s_i) + \hat{c}_k(r_x, s_i, f_i) \\ &\quad + \hat{c}_k(r_x, f_i, \hat{t}_{u+1}), \\ \hat{c}_k(r_x, s_i, f_i) &= \frac{f_i - s_i}{\hat{t}_{u+1} - \hat{t}_u} \hat{c}_1(r_x, \hat{t}_u, \hat{t}_{u+1}), \quad k = 1, 3 \\ \hat{c}_2(r_x, s_i, f_i) &= \frac{(f_i - s_i)^2}{(s_i - \hat{t}_u)^2 + (f_i - s_i)^2 + (\hat{t}_{u+1} - f_i)^2} \\ &\quad \hat{c}_2(r_x, \hat{t}_u, \hat{t}_{u+1}). \end{aligned} \quad (23)$$

$\hat{c}_k(r_x, \hat{t}_u, s_i)$ and $\hat{c}_k(r_x, f_i, \hat{t}_{u+1})$, $k = 1, 3$, can be computed similarly. Note that time interval $[\hat{t}_u, \hat{t}_{u+1}]$ has to be divided into three time intervals to match $[s_i, f_i]$.

2. If either $s_i = \hat{t}_u$ and $f_i < \hat{t}_{u+1}$ or $s_i < \hat{t}_u$ and $\hat{t}_u < f_i < \hat{t}_{u+1}$, $1 \leq u \leq \hat{\alpha} - 1$,

$$\begin{aligned} \hat{c}_k(r_x, \hat{t}_u, \hat{t}_{u+1}) &= \hat{c}_k(r_x, \hat{t}_u, f_i) + \hat{c}_k(r_x, f_i, \hat{t}_{u+1}), \\ \hat{c}_k(r_x, \hat{t}_u, f_i) &= \frac{f_i - \hat{t}_u}{\hat{t}_{u+1} - \hat{t}_u} \hat{c}_1(r_x, \hat{t}_u, \hat{t}_{u+1}), \quad k = 1, 3 \\ \hat{c}_2(r_x, \hat{t}_u, f_i) &= \frac{(f_i - \hat{t}_u)^2}{(f_i - \hat{t}_u)^2 + (\hat{t}_{u+1} - f_i)^2} \hat{c}_2(r_x, \hat{t}_u, \hat{t}_{u+1}). \end{aligned} \quad (24)$$

$\hat{c}_k(r_x, f_i, \hat{t}_{u+1})$, $k = 1, 2, 3$, can be computed similarly. Note that time interval $[\hat{t}_u, \hat{t}_{u+1}]$ has to be divided into two time intervals to match either $[s_i, f_i]$ or $[\hat{t}_u, f_i]$.

3. If either $s_i > \hat{t}_u$ and $f_i = \hat{t}_{u+1}$ or $s_i > \hat{t}_u$ and $f_i > \hat{t}_{u+1}$, $1 \leq u \leq \hat{\alpha} - 1$,

$$\begin{aligned} \hat{c}_k(r_x, \hat{t}_u, \hat{t}_{u+1}) &= \hat{c}_k(r_x, \hat{t}_u, s_i) + \hat{c}_k(r_x, s_i, \hat{t}_{u+1}), \\ \hat{c}_k(r_x, s_i, \hat{t}_{u+1}) &= \frac{\hat{t}_{u+1} - s_i}{\hat{t}_{u+1} - \hat{t}_u} \hat{c}_1(r_x, \hat{t}_u, \hat{t}_{u+1}), \quad k = 1, 3 \\ \hat{c}_2(r_x, s_i, \hat{t}_{u+1}) &= \frac{(\hat{t}_{u+1} - s_i)^2}{(s_i - \hat{t}_u)^2 + (\hat{t}_{u+1} - s_i)^2} \hat{c}_2(r_x, \hat{t}_u, \hat{t}_{u+1}). \end{aligned} \quad (25)$$

$\hat{c}_k(r_x, \hat{t}_u, s_i)$, $k = 1, 2, 3$, can be computed similarly. Note that time interval $[\hat{t}_u, \hat{t}_{u+1}]$ has to be divided into two time intervals to match either $[s_i, f_i]$ or $[s_i, \hat{t}_{u+1}]$.

After a matching and scheduling decision has been made, i.e., h_l is added to H , costs of using resource r_x in time intervals $[t_j, t_{j+1}]$, $1 \leq j \leq \alpha - 1$, must be updated as follows:

$$\hat{c}(r_x, t_j, t_{j+1}) = \begin{cases} \hat{c}(r_x, t_j, t_{j+1}), & \hat{c}(r_x, t_j, t_{j+1}) \geq c(r_x, t_j, t_{j+1}) \\ c(r_x, t_j, t_{j+1}), & \hat{c}(r_x, t_j, t_{j+1}) < c(r_x, t_j, t_{j+1}). \end{cases} \quad (26)$$

It should be noted that, if one of the special Cases 1-3 given above occurs, a time interval will be divided into either two or three subintervals and (26) will update the cost which

corresponds to only one of the subintervals of that time interval. Thus, the other(s) must be updated using (23)-(26), accordingly.

ACKNOWLEDGMENTS

A preliminary version of this paper was published in Proceedings of the 2000 International Conference on Parallel Processing (ICPP '00). Atakan Dogan is on leave from the Department of Electrical and Electronics Engineering, Anadolu University, Eskisehir, Turkey.

REFERENCES

- [1] R.F. Freund and H.J. Siegel, "Heterogeneous Processing," *Computer*, vol. 26, pp. 13-17, June 1993.
- [2] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.-L. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Computer*, vol. 26, pp. 18-27, June 1993.
- [3] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constraint Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 175-187, Feb. 1993.
- [4] L. Wang, H.J. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *J. Parallel and Distributed Computing*, vol. 47, pp. 8-22, Nov. 1997.
- [5] M.A. Iverson and F. Özgüner, "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment," *Proc. 1998 Workshop Heterogeneous Processing*, pp. 70-78, Mar. 1998.
- [6] B.R. Carter, D.W. Watson, R.F. Freund, E. Keith, F. Mirabile, and H.J. Siegel, "Generational Scheduling for Dynamic Task Management in Heterogeneous Computing Systems," *Information Science*, vol. 106, no. 3-4, pp. 219-236, 1998.
- [7] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. 1998 Workshop Heterogeneous Processing*, pp. 57-69, Mar. 1998.
- [8] S.M. Shatz, J.P. Wang, and M. Goto, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," *IEEE Trans. Computers*, vol. 41, pp. 1156-1168, Sept. 1992.
- [9] S. Kartik and C.S.R. Murthy, "Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems," *IEEE Trans. Computers*, vol. 46, pp. 719-724, June 1997.
- [10] S.M. Shatz and J.P. Wang, "Models & Algorithms for Reliability-Oriented Task-Allocation in Redundant Distributed-Computer Systems," *IEEE Trans. Reliability*, vol. 38, pp. 16-26, Apr. 1989.
- [11] S. Kartik and C.S.R. Murthy, "Improved Task Allocation Algorithms to Maximize Reliability of Redundant Distributed Computing Systems," *IEEE Trans. Reliability*, vol. 44, pp. 575-586, Dec. 1995.
- [12] A. Dogan and F. Özgüner, "Reliable Scheduling of Precedence-Constrained Tasks Using a Genetic Algorithm," *Proc. 2000 Int'l Conf. Parallel and Distributed Processing Techniques and Application*, pp. 549-555, June 2000.
- [13] A. Dogan and F. Özgüner, "Optimal and Suboptimal Reliable Scheduling of Precedence-Constrained Tasks in Heterogeneous Computing," *Proc. 2000 Int'l Conf. Parallel Processing Workshop Network Based Computing*, pp. 429-436, Aug. 2000.
- [14] M.A. Iverson, "Dynamic Mapping and Scheduling Algorithms for a Multi-User Heterogeneous Computing Environment," PhD thesis, Ohio State Univ., Columbus, 1999.
- [15] M.O. Ball, "Computational Complexity of Network Reliability Analysis: An Overview," *IEEE Trans. Reliability*, vol. 35, pp. 230-239, Aug. 1986.
- [16] S. Rai and K.K. Aggarwal, "An Efficient Method for Reliability Evaluation of a General Network," *IEEE Trans. Reliability*, vol. 27, pp. 206-211, Aug. 1978.
- [17] C.S. Raghavendra and S.V. Makam, "Reliability Modeling and Analysis of Computer Networks," *IEEE Trans. Reliability*, vol. 35, pp. 156-160, June 1986.
- [18] P.A. Jensen and M. Bellmore, "An Algorithm to Determine the Reliability of a Complex System," *IEEE Trans. Reliability*, vol. 18, pp. 169-174, Nov. 1969.
- [19] Y.G. Chen and M.C. Yuang, "A Cut-Based Method for Terminal-Pair Reliability," *IEEE Trans. Reliability*, vol. 45, pp. 413-416, Sept. 1996.
- [20] J.S. Plank and W.R. Elwasif, "Experimental Assessment of Workstation Failures and Their Impact on Checkpointing Systems," *Int'l Symp. Fault-Tolerant Computing*, pp. 48-57, June 1998.
- [21] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. MIT Press, 1997.
- [22] A. Dogan and F. Özgüner, "Trading Off Execution Time for Reliability in Scheduling Precedence-Constrained Tasks in Heterogeneous Computing," *Proc. Int'l Parallel and Distributed Processing Symposium*, Apr. 2001.
- [23] E.E. Lewis, *Introduction to Reliability Engineering*. John Wiley & Sons, 1987.
- [24] Interagency Working Group on Information Technology Research and Development, "Information Technology: The 21st Century Revolution," *FY2001 Blue Book*, Sept. 2000.
- [25] M.A. Iverson, F. Özgüner, and G.J. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," *Proc. 1995 Workshop Heterogeneous Processing*, pp. 93-100, 1995.
- [26] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.
- [27] R. Wolski, N.T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *J. Future Generation Computing Systems*, 1999.
- [28] M.A. Iverson, F. Özgüner, and L. Potter, "Statistical Prediction of Task Execution Times through Analytic Benchmarking for Scheduling in a Heterogeneous Environment," *IEEE Trans. Computers*, vol. 48, Dec. 1999.



Atakan Dogan received the BS degree and the MS degree in electrical and electronics engineering from Anadolu University, Eskisehir, Turkey, in 1995 and 1997, respectively. He is currently a PhD candidate in electrical engineering at the Ohio State University. His research interests include heterogeneous distributed computing and interprocessor communication on parallel and distributed memory multiprocessor systems. He is a student member of the IEEE.



Füsün Özgüner received the MS degree in electrical engineering from the Istanbul Technical University in 1972 and the PhD degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1975. She worked at the IBM T.J. Watson Research Center with the Design Automation group for one year and joined the faculty at the Department of Electrical Engineering, Istanbul Technical University in 1976. Since January 1981, she has been with the Ohio State University, where she is presently a professor of electrical engineering. She also serves as the director for research computing at the Office of the Chief Information Officer. Her current research interests are parallel and fault-tolerant architectures, heterogeneous computing, reconfiguration and communication in parallel architectures, real-time parallel computing and communication, wireless networks, and parallel algorithm design. She has served as an associate editor of *IEEE Transactions on Computers* and on program committees of several international conferences. She is a member of the IEEE.

► For more information on this or any computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.