Task Scheduling for Maximizing Performance and Reliability Considering Fault Recovery in Heterogeneous Distributed Systems

Chi-Yeh Chen

Abstract—Machine and network failures worsen the results of executing applications on system. Therefore, the reliability of applications on system is an important issue. The recovery of failed machines may increase processing time. This work studies the expected makespan to schedule tasks in heterogeneous distributed systems. A task may be replicated many times to reduce the expected execution time. A two-phase algorithm is proposed. The first phase uses a linear program formulation and a rounding procedure to obtain a favorable allotment to minimize the expected makespan. The second phase applies a scheduling method that is based on the expected executed time and the communication time. During execution, two strategies are considered. In the first strategy, no replication of a task on a set of processors can be stopped. In the second strategy, once a replication of a task has been completed, the other replications of the task are immediately aborted. A comparison reveals that the proposed algorithm significantly outperformed previously proposed algorithms in terms of schedule length ratio, reliability and speedup.

Index Terms—Scheduling algorithm, heterogeneous systems, reliability, precedence constraints

1 Introduction

As technology advances, powerful computers and high-speed networks become more widely used. Large-scale high-performance distributed computing environments, or computational grids can be easily established, but they give rise to problems concerning quality of service (QoS) targets, including reliability, performance and throughput. An important challenge in efficiently using a parallel and distributed system is to develop task scheduling algorithms. Scheduling algorithms minimize makespan and this problem has been proven to be $\mathcal{NP}$-Complete [10]. Therefore, solving this problem in polynomial time is difficult and efficient approximation heuristics are sought. In large-scale systems, machine and network failures worsen the effectiveness of the executed applications. Therefore, the reliability of applications is an important problem. Recently, many studies have addressed bi-criteria (makespan and reliability) multiprocessor scheduling [2], [7], [8], [11], [14], [22], [23]. The active replication of task and data communication is a common means of increasing the reliability of a system. Intuitively, adding replications increases not only reliability but also, in general, the schedule length [11].

The *Local Node Fault Recovery* (LNFR) mechanism supports fault recovery in grid systems. If a processor fails, the failed processor is repaired or replaced; and once it becomes operational again, task execution may be resumed by a recovery action. The failures in grid systems can be classified into two categories—unrecoverable and recoverable. Software failures that are caused by embedded faults in programs are unrecoverable. Since no recovering module can be installed on communication links, communication link failures are unrecoverable. Hardware failures can be unrecoverable or recoverable. The recoverable failures are caused by human error or performance overload. An unrecoverable hardware failure will results in the termination of the current task [13].

An application can be represented on a directed acyclic graph (DAG) in which the nodes represent tasks and the edges represent precedence constraints among tasks. Many effective heuristic scheduling algorithms have been proposed for DAG scheduling, including the mapping heuristic (MH) [9], the dynamic-level scheduling (DLS) algorithm [25], the levelized-min time (LMT) algorithm [18], the Critical-Path-on-a-Machine (CPOP) algorithm [30], the heterogeneous earliest-finish-time (HEFT) algorithm [28], [30] and the Dominant Sequence Clustering (DSC) algorithm [31]. Recently, a few algorithms have been proposed to minimize schedule length and increase reliability. He et al. [17] proposed two algorithms use in embedded heterogeneous systems. Tang et al. [27] proposed a reliability-aware scheduling algorithm (RASD) for small-scale heterogeneous distributed systems. Tang et al. [29] presented a Hierarchical Reliability-Driven Scheduling algorithm (HRDS) in a grid computing system.

Ahmad and Kwok [1] proposed a technique that systematically decomposes the task graph into critical path node, in-branch node, and out-branch node bindings. These bindings help to determine the relative importance of nodes. The performance can be improved using the task duplication technique, which reduces the start time of important nodes. Girault et al. [12] presented a two-step heuristic for optimizing both makespan and reliability, which adds

• *The author is with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Road, Tainan, Taiwan, ROC. E-mail: chency@csie.ncku.edu.tw.*

Manuscript received 24 July 2014; revised 3 Nov. 2014; accepted 9 Feb. 2015. Date of publication 12 Feb. 2015; date of current version 20 Jan. 2016. Recommended for acceptance by D. Trystram.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2403861

random replications to reach a reliability threshold, and then computes the temporal allocation function to minimize the makespan. Jeannot et al. [20] proposed an optimal scheduling algorithm whose objective is to maximize the reliability subject to makespan minimization; they also provided a $(1+\epsilon,1)$-approximation algorithm of the Pareto-front. For independent arbitrary tasks, they proposed a $(2,1)$-approximation algorithm. Hashimoto et al. [15], [16] proposed a scheduling algorithm to realize fault tolerance in multiprocessor systems. This algorithm partitions a task graph into subsets of tasks, based on the height of a task graph. For each subset, the algorithm successively duplicates and the schedules the tasks in the subset. Bouguerra et al. [4] presented bi-objective scheduling algorithms for optimizing application makespan and reliability when failure rates are increasing or decreasing over time.

The recovery of failures may extend the processing time. A schedule without the replication of a task may have a short makespan if no failure occurs, but with a long expected makespan. The expected makespan is reduced by properly increasing the number of replications. This work concentrates on the expected makespan. The proposed algorithm is a two-phase algorithm. The first phase finds a favorable number of replication of tasks. The second phase applies a scheduling method that is based on the expected executed time and the communication time.

A replication problem is herein transformed into the problem of allocating malleable tasks and the features of the expected processing time and the expected work associated with a task are analyzed, where the latter equals the total expected processing time in a multiprocessor system. An allocation method, which uses a linear program formulation and a rounding procedure to find a feasible allotment to minimize the expected makespan, is presented. A method for evaluating the expected finish time of each task and to schedule all tasks, is also developed. The experimental results reveal that the proposed algorithm significantly outperforms previously developed methods in terms of schedule length ratio (SLR), reliability and speedup.

The rest of this paper is organized as follows. Section 2 introduces basic definitions, notation, and assumptions. Section 3 presents the allocation method. Section 4 presents the scheduling algorithm. Section 5 summarizes experimental results concerning the performance and reliability. Section 6 draws conclusions.

2 Assumptions and Notation

This work considers a heterogeneous distributed system that has many computing resources with various processing capabilities. The topology of the distributed system is modeled as an undirected graph $GT=(P,L)$ where P represents a set of m processors, that are assumed to be fully linked by a set of links L. An edge $l_{i,j}\in L$ is a bidirectional communication link between the incident processors p_i and p_j. The computation capacity refers to the number of operations per unit time, namely, the weight $w(p_i)$ of processor $p_i\in P$. The communication capacity refers to the data transfer rate, namely, the weight $w(l_{i,j})$ of edge $l_{i,j}$. If two tasks are scheduled on a single processor, then the communication cost between the two tasks is zero. For convenience of specification, let $l_{i,i}$ be the internal communication link in the processor p_i and the weight $w(l_{i,i})=\infty$.

The application is represented as a set of generic tasks, each linked to others by *precedence constraints*. The precedence constraints are first determined by analyzing the data flow among the tasks. Let $G=(V,E,w)$ be a directed acyclic graph, where $V=\{v_1,v_2,\ldots,v_n\}$ represents the set of tasks and $E\subseteq V\times V$ is the set of precedence constraints among the tasks. In a situation in which an arc $e_{i,j}=(i,j)\in E$, task v_i must be completely processed before a task v_j can be executed. Task v_i is called a predecessor of v_j, and task v_j is a successor of v_i. The weight $w(v_i)$ of task v_i is its computational cost and the weight $w(e_{i,j})$ of edge $e_{i,j}$ is its communication cost. Let S_i be the set of processors that are allocated to task v_i. The processing time of v_i on processor p_x is denoted as $t_p(v_i,p_x)=w(v_i)/w(p_x)$. The communication time of $e_{i,j}$ on link $l_{k,x}$ is denoted as $t_c(e_{i,j},l_{k,x})=w(e_{i,j})/w(l_{k,x})$. The communication time of an edge depends on the source processor and the destination processor.

Definition 1. *The communication time of $e_{i,j}$ on link $l_{k,x}$ is given by $t_c(e_{i,j},l_{k,x})=0$ if $k=x$, otherwise $t_c(e_{i,j},l_{k,x})=w(e_{i,j})/w(l_{k,x})$.*

The set of direct predecessors of task v_i is represented as $\mathbf{pred}(v_i)$ and the set of direct successors of v_i is represented as $\mathbf{succ}(v_i)$. A task v_i without any predecessor is called an entry task and a task without any successor is called an exit task. Without loss of generality, the DAG is assumed to have only one entry task v_{entry} and only one exit task v_{exit}. If multiple exit tasks or entry tasks exist, they may be connected with zero time-weight edges to a single pseudo-exit task or a single pseudo-entry task that has zero time-weight. Table 1 presents the notation and terminology that are used herein.

2.1 Analysis of Task Reliability

This work uses the widely accepted reliability model of Shatz and Wang [24] that the failure of processors and communication links in a system is assumed to follow a Poisson process and each resource is characterized by a constant failure rate per unit time λ. Restated, the reliability of a resource during the interval d is $e^{-\lambda d}$. The failure rate of processor p_i is denoted as λ_i and the failure rate of link $l_{i,j}$ is denoted as $\lambda_{l_{i,j}}$. The internal communication in a processor is considered to be failure-free, so $\lambda_{l_{i,i}}=0$. Plank and Elwasif [21] demonstrate that modeling the failure of a resource by a Poisson process is a useful assumption. A failure on processor p_i is recoverable with a probability of γ_i.

The probability that communication is reliable and the probability that it fails between v_i and v_j are

$$R[E_{e_{i,j},p_k,p_x}]=e^{-\lambda_{l_{k,x}}t_c(e_{i,j},l_{k,x})}$$

and

$$F[E_{e_{i,j},p_k,p_x}]=1-R[E_{e_{i,j},p_k,p_x}],$$

respectively, where task v_i is executed on processor p_k and the task v_j is executed on processor p_x. The reliability

TABLE 1
Notation and Terminology

P, L	A set of m processors and a set of links.
p_i, $l_{i,j}$	A processor and a bidirectional communication link between the incident processors p_i and p_j.
$w(p_i)$	The weight of processor $p_i \in P$ (the number of operations per unit time.)
$w(l_{i,j})$	The weight of edge $l_{i,j}$ (the data transfer rate.)
V, E	The set of tasks and the set of precedence constraints among the tasks.
$w(v_i)$	The weight of task v_i (computational cost.)
$w(e_{i,j})$	The weight of edge $e_{i,j}$ (communication cost.)
S_i	The set of processors that is allocated to task v_i.
$t_p(v_i, p_x)$	The processing time of v_i on processor p_x, namely $t_p(v_i, p_x) = w(v_i)/w(p_x)$.
$t_c(e_{i,j}, l_{k,x})$	The communication time of $e_{i,j}$ on link $l_{k,x}$, namely $t_c(e_{i,j}, l_{k,x}) = w(e_{i,j})/w(l_{k,x})$.
$\mathbf{pred}(v_i)$	The set of direct predecessors of task v_i.
$\mathbf{succ}(v_i)$	The set of direct successors of v_i.
λ_i	The failure rate of processor p_i.
$\lambda_{l_{i,j}}$	The failure rate of link $l_{i,j}$.
γ_i	The probability that a failure on processor p_i is recoverable.
$R[E_{e_{i,j},p_k,p_x}]$	The communication reliability probability between v_i and v_j on processor p_k and processor p_x respectively.
$F[E_{e_{i,j},p_k,p_x}]$	The communication communication failure probability between v_i and v_j, namely $1 - R[E_{e_{i,j},p_k,p_x}]$.
$R[E_{v_i,p_x}]$	The reliability probability of v_i on processor p_x.
$F[E_{v_i,p_x}]$	The failure probability of v_i on processor p_x.
$\mathbf{\Phi}[E_{v_i,p_x}]$	The expected overhead of task v_i on processor p_x.
$\bar{t}_p(v_i)$	The mean processing time of a task v_i.
$\bar{t}_c(e_{i,j})$	The mean communication time of edge $e_{i,j}$.
$\bar{R}[E_{v_j}]$	The mean reliability of task v_j.
$\bar{R}[E_{e_{i,j}}]$	The mean communication reliability probability between v_i and v_j.
$\bar{\Phi}[E_{v_i}]$	The mean overhead of task v_i on processor p_x.
F_i	The mean failed probability of task v_i, namely $F_i = 1 - \bar{R}[E_{v_i}]$.
$T_p(v_i, \alpha_i)$	The expected processing time of the task v_i that executes on α_i processors.
$W(v_i, \alpha_i)$	The expected work of task v_i that executes on α_i processors.
$AL(i)$	The lower bound on the number of allotted processors for task v_i.
β	The weight of reliability overhead.
$\mathrm{EST}(v_i, p_k)$	The earliest start time of task v_i on a processor p_k.
$\mathrm{EFT}(v_i, p_k, S_i)$	The earliest finish time of task v_i on processor p_k with other replication of tasks on a set of processors S_i.
$\mathrm{EFT}(e_{i,j}, S_i, p_x)$	The earliest communication finish time of edge $e_{i,j}$ from $p_k \in S_i$ to p_x.

probability of v_i and the failure probability of v_i on processor p_x are

$$R[E_{v_i,p_x}] = e^{-\lambda_x t_p(v_i,p_x)}$$

and

$$F[E_{v_i,p_x}] = 1 - R[E_{v_i,p_x}],$$

respectively.

The time of failure during the execution of a resource is assumed to be uniformly distributed. When a resource fails, the expected overhead of task v_i on processor p_x is

$$\mathbf{\Phi}[E_{v_i,p_x}] = \int_0^{t_p(v_i,p_x)} \frac{t}{t_p(v_i,p_x)} dt + \theta_{p_x} = \frac{t_p(v_i,p_x)}{2} + \theta_{p_x},$$

where θ_{p_x} is the recovery time of processor p_x.

2.2 Normalization Step

A problem for scheduling tasks on a heterogeneous computing system is that the processing time of tasks on a processor may differ from those of the same tasks on another processor. Therefore, the priority of tasks for the scheduling algorithm cannot be obtained. One method for solving this problem is to normalize a heterogeneous system into a homogenous system. The DLS algorithm and the HEFT algorithm set the computation costs of tasks to their median values and their mean values, respectively. In this work, the mean computational cost, mean communication cost and mean reliability are used to solve the allotment problem and to compute the priorities of tasks. For a task v_i on a heterogeneous computing system with m processors, the mean processing time is computed using

$$\bar{t}_p(v_i) = \sum_{x=1}^{m} \frac{w(v_i)}{m w(p_x)}.$$

The mean communication time of edge $e_{i,j}$ is computed using

$$\bar{t}_c(e_{i,j}) = \sum_{l_{x,k} \in L} \frac{w(e_{i,j})}{|L| w(l_{x,k})}.$$

The mean reliability of task v_j is computed using

$$\bar{R}[E_{v_j}] = \sum_{x=1}^{m} \frac{R[E_{v_j,p_x}]}{m}.$$

The mean communication reliability probability between v_i and v_j is computed using

$$\bar{R}[E_{e_{i,j}}] = \sum_{k_1=1}^{m} \sum_{k_2=k_1}^{m} \frac{2R[E_{e_{i,j},p_{k_1},p_{k_2}}]}{m(m-1)}.$$

The mean overhead of task v_i on processor p_x is

$$\bar{\Phi}[E_{v_i}] = \int_0^{\bar{t}_p(v_i)} \frac{t}{\bar{t}_p(v_i)} dt + \theta_p = \frac{\bar{t}_p(v_i)}{2} + \theta_p,$$

where θ_p is the mean recovery time of the processor.

3 Allotment Problem

The recovery of failures may delay completion. Accordingly, this work uses a replication method to reduce the delay. This replication method executes many the same task on many processors. Two strategies, *Strategy I* and *Strategy II*, are considered. In Strategy I, no replication of a task can be stopped. In Strategy II, once a replication of a task is completed, the other replications of the task are immediately terminated. Restated, only one replication of a task is completed. This section solves the allotment problem for $G = (V, E, w)$ without communication cost ($w(e_{i,j}) = 0$) in a homogenous system. The execution of a task is assume to have to be restarted on a failed processor after the processor has been recovered. Let E_p be the event that the failure of p processors. The event that p processors successively fail i times is denoted as $E_{p,i}$. The expected processing time of task v_i that is executed on α_i processors is

$$\begin{aligned} T_p(v_i, \alpha_i) &= \bar{t}_p(v_i) + \bar{\Phi}[E_{v_i}] \sum_{k=1}^{\infty} \mathbb{P}(E_{\alpha_i} | E_{\alpha_i, k-1}) \\ &= \bar{t}_p(v_i) + \left(\frac{\bar{t}_p(v_i)}{2} + \theta_p\right) \sum_{k=1}^{\infty} F_i^{k\alpha_i} \\ &= \bar{t}_p(v_i) + \frac{F_i^{\alpha_i}}{1 - F_i^{\alpha_i}} \left(\frac{\bar{t}_p(v_i)}{2} + \theta_p\right), \end{aligned} \quad (1)$$

where $F_i = 1 - \bar{R}[E_{v_i}]$ is the mean probability of failure of task v_i. When the failure probability $F_i = 0$, the problem is a general scheduling problem. When $F_i = 1$, the expected processing time is infinite. Herein, $0 < F_i < 1$ is assumed. From the equation (1), the following theorem can be obtained.

Theorem 3.1. *The expected processing time $T_p(v_i, \alpha_i)$ is strictly decreasing in the number α_i of allocated processors: $T_p(v_i, \alpha_i) \leq T_p(v_i, \alpha_i')$, for $\alpha_i \geq \alpha_i'$.*

3.1 Strategy I

Let $B(\alpha_i, F_i)$ be a binomially distributed random variable with parameters α_i and F_i. The work of a task is the total execution time of all replicated tasks, which corresponds to the total usage of computation resources. In Strategy I, the expected work of task v_i is computed as

$$\begin{aligned} W(v_i, \alpha_i) &= \alpha_i \bar{t}_p(v_i) + \bar{\Phi}[E_{v_i}] g(\alpha_i) \\ &= \alpha_i \bar{t}_p(v_i) + \left(\frac{\bar{t}_p(v_i)}{2} + \theta_p\right) \alpha_i \sum_{k=1}^{\infty} F_i^k \\ &= \alpha_i \bar{t}_p(v_i) + \frac{\alpha_i F_i}{1 - F_i} \left(\frac{\bar{t}_p(v_i)}{2} + \theta_p\right), \end{aligned} \quad (2)$$

where $g(\alpha_i) = \sum_{k=1}^{\alpha_i} \mathbb{P}(B(\alpha_i, F_i) = k)(k + g(k))$. From the equation (2), the following theorems can be obtained.

Theorem 3.2. *In Strategy I, the work $W(v_i, \alpha_i)$ is non-decreasing in the number α_i of allocated processors: $W(v_i, \alpha_i) \leq W(v_i, \alpha_i')$ for $\alpha_i \leq \alpha_i'$.*

Theorem 3.3. *In Strategy I, the work $W(v_i, \alpha_i)$ of a task is convex in the expected processing time.*

Proof. A function $f : [a, b] \to \mathbb{R}$ is convex if and only if the slope of the function is monotonically non-decreasing in $[a, b]$. That

$$\frac{W(v_i, \alpha_i + 1) - W(v_i, \alpha_i)}{T_p(v_i, \alpha_i + 1) - T_p(v_i, \alpha_i)} \geq \frac{W(v_i, \alpha_i + 2) - W(v_i, \alpha_i + 1)}{T_p(v_i, \alpha_i + 2) - T_p(v_i, \alpha_i + 1)}$$

for $\alpha_i \geq 1$ which is equivalent to $0 \leq 1 + F_i^{\alpha_i + 1}$, must be proven. Since $0 < F_i < 1$, the proof is complete. □

3.2 Strategy II

In Strategy II, the expected work of task v_i is computed as

$$W(v_i, \alpha_i) = \alpha_i \bar{t}_p(v_i) + \frac{\alpha_i F_i^{\alpha_i}}{1 - F_i^{\alpha_i}} \left(\frac{\bar{t}_p(v_i)}{2} + \theta_p\right). \quad (3)$$

Define

$$f_1(v_i, \alpha_i) = \frac{2\left(F_i^{2\alpha_i + 1} - (\alpha_i + 1)F_i^{\alpha_i + 1} + \alpha_i F_i^{\alpha_i}\right)}{2 + F_i^{2\alpha_i + 1} + (\alpha_i - 1)F_i^{\alpha_i + 1} - (\alpha_i + 2)F_i^{\alpha_i}}$$

and

$$f_2(v_i, \alpha_i) = \frac{2F_i^{\alpha_i + 1}\left(1 - F_i^{\alpha_i + 1}\right)}{2 - F_i^{2\alpha_i + 2} - F_i^{\alpha_i + 1}}.$$

From the equation (3), the following theorems can be obtained.

Lemma 3.4. *In Strategy II, $\frac{\bar{t}_p(v_i)}{\theta_p} \geq f_1(v_i, \alpha_i)$ if and only if $W(v_i, \alpha_i) \leq W(v_i, \alpha_i + 1)$.*

Proof (Necessary). Suppose, by contradiction, that $W(v_i, \alpha_i) > W(v_i, \alpha_i + 1)$ which is equivalent to

$$\begin{aligned} &\bar{t}_p(v_i)\left(2 + F_i^{2\alpha_i + 1} + (\alpha_i - 1)F_i^{\alpha_i + 1} - (\alpha_i + 2)F_i^{\alpha_i}\right) \\ &< 2\theta_p\left(F_i^{2\alpha_i + 1} - (\alpha_i + 1)F_i^{\alpha_i + 1} + \alpha_i F_i^{\alpha_i}\right). \end{aligned}$$

Since θ_p and $2 + F_i^{2\alpha_i + 1} + (\alpha_i - 1)F_i^{\alpha_i + 1} - (\alpha_i + 2)F_i^{\alpha_i}$ are positive in $0 < F_i < 1$ and $\alpha_i \geq 1$, $\frac{\bar{t}_p(v_i)}{\theta_p} < f_1(v_i, \alpha_i)$. This completes the proof of the contrapositive.

Sufficiency. We prove the contrapositive. Suppose $\frac{\bar{t}_p(v_i)}{\theta_p} < f_1(v_i, \alpha_i)$. Similarly to the necessary part, $W(v_i, \alpha_i) > W(v_i, \alpha_i + 1)$ can be obtained. □

Theorem 3.5. *In Strategy II, if $\frac{\bar{t}_p(v_i)}{\theta_p} \geq f_1(v_i, \alpha_i)$ for some $\alpha_i \geq 1$, $W(v_i, \alpha_i') \leq W(v_i, \alpha_i' + 1)$ for all $\alpha_i' \geq \alpha_i$.*

Proof. Since $f_1(v_i, \alpha_i)$ is non-increasing in $\alpha_i \geq 1$, $\frac{\bar{t}_p(v_i)}{\theta_p} \geq f_1(v_i, \alpha_i')$ is fulfilled for all $\alpha_i' \geq \alpha_i$. According to Lemma 3.4, $W(v_i, \alpha_i') \leq W(v_i, \alpha_i' + 1)$ for all $\alpha_i' \geq \alpha_i$. □

Lemma 3.6. *In Strategy II, $\frac{\bar{t}_p(v_i)}{\theta_p} \geq f_2(v_i, \alpha_i)$ if and only if the work $W(v_i, \alpha_i')$ is convex in $[T_p(v_i, \alpha_i + 2), T_p(v_i, \alpha_i)]$ for $\alpha_i \leq \alpha_i' \leq \alpha_i + 2$.*

Proof (Necessary). Suppose, by contradiction, that the work $W(v_i,\alpha_i')$ is not convex in $[T_p(v_i,\alpha_i+2),T_p(v_i,\alpha_i)]$ for $\alpha_i \le \alpha_i' \le \alpha_i+2$ which is equivalent to

$$\bar{t}_p(v_i)\Big(2-F_i^{2\alpha_i+2}-F_i^{\alpha_i+1}\Big) < 2\theta_p F_i^{\alpha_i+1}\Big(1-F_i^{\alpha_i+1}\Big).$$

Since θ_p and $2-F_i^{2\alpha_i+2}-F_i^{\alpha_i+1}$ are positive for $0<F_i<1$ and $\alpha_i \ge 1$, $\frac{\bar{t}_p(v_i)}{\theta_p} < f_2(v_i,\alpha_i)$. This completes the proof of the contrapositive.

Sufficiency. We prove the contrapositive. Suppose $\frac{\bar{t}_p(v_i)}{\theta_p} < f_2(v_i,\alpha_i)$. Similarly to the necessary part,

$$\frac{W(v_i,\alpha_i+1)-W(v_i,\alpha_i)}{T_p(v_i,\alpha_i+1)-T_p(v_i,\alpha_i)} < \frac{W(v_i,\alpha_i+2)-W(v_i,\alpha_i+1)}{T_p(v_i,\alpha_i+2)-T_p(v_i,\alpha_i+1)}$$

can be obtained. □

Theorem 3.7. *In Strategy II, if* $\frac{\bar{t}_p(v_i)}{\theta_p} \ge f_2(v_i,j)$ *for some* $j \ge 1$, *the work* $W(v_i,j')$ *is convex in* $[T_p(v_i,m),T_p(v_i,j)]$ *for* $j \le j' \le m$.

Proof. Since $f_2(v_i,j)$ is non-increasing in $j \ge 1$, $\frac{\bar{t}_p(v_i)}{\theta_p} \ge f_2(v_i,j')$ is fulfilled for all $j \le j' \le m$. According to Lemma 3.6, the work $W(v_i,j')$ is convex in $[T_p(v_i,m), T_p(v_i,j)]$ for $j \le j' \le m$. □

Theorem 3.8. *In Strategy II, if* $\frac{\bar{t}_p(v_i)}{\theta_p} \ge f_1(v_i,\alpha_i)$ *for some* $\alpha_i \ge 1$, *then the work* $W(v_i,\alpha_i')$ *is convex in* $[T_p(v_i,\alpha_i+2),T_p(v_i,\alpha_i)]$ *for* $\alpha_i \le \alpha_i' \le \alpha_i+2$.

Proof. We need to prove that $f_1(v_i,\alpha_i) \ge f_2(v_i,\alpha_i)$ which is equivlent to

$$2\Big(\alpha_i-2F_i-\alpha_iF_i+2F_i^{\alpha_i+1}+\alpha_iF_i^{\alpha_i+1}-\alpha_iF_i^{\alpha_i+2}\Big)\Big(1-F_1^{\alpha_i+1}\Big) \ge 0.$$

Since $2(1-F_1^{\alpha_i+1}) \ge 0$, we just need to prove $\alpha_i-2F_i-\alpha_iF_i+2F_i^{\alpha_i+1}+\alpha_iF_i^{\alpha_i+1}-\alpha_iF_i^{\alpha_i+2} \ge 0$ which is equivlent to

$$\frac{\alpha_i\Big(1+F_i^{\alpha_i+1}\Big)(1-F_i)}{F_i\big(1-F_i^{\alpha_i}\big)} \ge 2.$$

Since the slope of the left hand side is positive in $0<F_i<1$ and $\alpha_i \ge 1$, it is increasing in $\alpha_i \ge 1$. Thus, $\big(1+F_i^2\big)/F_i \ge 2$ needs to prove. Solving $\frac{d}{dF_i}(1+F_i^2)/F_i = 0$, the roots are $F_i = \pm 1$ and $(1+F_i^2)/F_i$ has a minimum value of 2 at $F_i = 1$. The inequality $\big(1+F_i^2\big)/F_i \ge 2$ is proved. □

3.3 Solving Allotment Problem

According to the analysis in Sections 3.1 and 3.2, the tasks can be mapped as malleable tasks. Accordingly, this work modifies a linear program in [5], [19] to solve the allotment problem. In Strategy II, if $f_1(v_i,\alpha_i+1) \le \frac{\bar{t}_p(v_i)}{\theta_p} < f_1(v_i,\alpha_i)$ for some $\alpha_i \ge 1$, then the expected processing time $T_p(v_i,\alpha_i')$ and the expected work $W(v_i,\alpha_i')$ are strictly decreasing in $1 \le \alpha_i' \le \alpha_i+1$. According to Theorems 3.5, 3.7 and 3.8, the expected work $W(v_i,\alpha_i')$ is non-decreasing in $\alpha_i+1 \le \alpha_i' \le m$ and is convex in $[T_p(v_i,m),T_p(v_i,\alpha_i+1)]$. An optimal solution for allotment of task v_i is between α_i and m. Let $AL(i)$ be the lower bound on the number of allotted processors for task v_i. The algorithm of allotment lower bound (*Algorithm 1*) determines the value of $AL(i)$. Let $V_s = \{v_i \in V | \frac{\bar{t}_p(v_i)}{\theta_p} < f_1(v_i,m-1)\}$ be the set of small tasks and $V_e = \{v_i \in V | \mathbf{pred}(v_i) = \emptyset\}$ be the set of entry tasks. The following linear program for solving allotment problem, based on [5], [19], must be solved.

Algorithm 1. Allorment Lower Bound

```
1:  In the Strategy I
2:  AL(i) = 1 for all 1 ≤ i ≤ n.
3:  In the Strategy II
4:  for i ← 1 to n do
5:    if t̄_p(v_i)/θ_p ≥ f_1(v_i, 1) then
6:      AL(i) = 1.
7:    else if t̄_p(v_i)/θ_p < f_1(v_i, m − 1) then
8:      AL(i) = m.
9:    else
10:     Find a number j ∈ [1, m − 2] such that
        f_1(v_i, j + 1) ≤ t̄_p(v_i)/θ_p < f_1(v_i, j).
11:     AL(i) = j + 1.
```

$$\begin{array}{lll}
\min & C & \\
s.t. & 0 \le C_i \le L, & \forall v_i \in V; \\
& C_i + x_k \le C_k, & \forall v_i \in V, v_k \in \mathbf{succ}(v_i); \\
& C_i = x_i, & \forall v_i \in V_e; \\
& x_i \in [T_p(v_i,m), T_p(v_i, AL(i))], & \forall v_i \in V; \\
& \bar{\omega}_i(x_i,j) \le \hat{w}_i, & \forall v_i \in V \backslash V_s, j \in [AL(i), m-1]; \\
& \hat{w}_i = W(v_i,m), & \forall v_i \in V_s; \\
& L \le C; & \\
& W/m = \sum_{i=1}^{n} \hat{w}_i/m \le C. &
\end{array}$$

The work function for the fractional solution is as follows:

$$\bar{\omega}_i(x_i^*,j) = W(v_i,j) + \frac{x_i^* - T_p(v_i,j)}{T_p(v_i,j+1)-T_p(v_i,j)}\big(W(v_i,j+1)-W(v_i,j)\big)$$

for all $v_i \in V \setminus V_s$.

The rounding procedure is applied to a fractional solution to yield a rounded solution, as follows. Suppose that the optimal solution $x_i^* \in [T_p(v_i,j+1),T_p(v_i,j)]$. If $x_i^* - T_p(v_i,j+1) < \rho \cdot (T_p(v_i,j)-T_p(v_i,j+1))$, then x_i^* is rounded to $x_i = T_p(v_i,j+1)$; otherwise, x_i^* is rounded to $x_i = T_p(v_i,j)$, where $\rho \in [0,1]$ is a rounding parameter. Therefore, the rounded solution $x_i \in \{T_p(v_j,m),\ldots,T_p(v_j,AL(i))\}$. Allotted processors α_i' can be identified such that $T_p(v_i,\alpha_i') = x_i$, yielding an allotment α' such that α_i' processors are allotted to job $v_i \in V$.

According to the analysis in [5], the allotment parameter is

$$\mu = \frac{m(2+\rho) - \sqrt{(2+2\rho+\rho^2)m^2 - 2m(1+\rho)}}{2}.$$

TABLE 2
[5] The Rounding Parameter ρ for $2 \leq m \leq 13$

m	$\rho(m)$	m	$\rho(m)$
2	0	8	0.5921
3	0.1198	9	0.6703
4	0.2290	10	0.7450
5	0.3293	11	0.8164
6	0.4225	12	0.8851
7	0.5098	13	0.9513

The rounding parameter is

$$\rho = 0.9999, \text{for } m \geq 14$$

and the value of ρ for $2 \leq m \leq 13$ is as given in Table 2.

The reliability-aware rank (RRank) attribute based on expected value is used herein to compute the priorities of tasks. The RRank is defined by Definition 2.

Definition 2. *Given an application DAG with n tasks and $|E|$ edges, and a heterogeneous system with m processors and $m(m-1)/2$ communication links, the RRank in a particular scheduling step is a rank of a task that equals the sum of the expected computation time and the maximum value of the RRank of the successors with the communication time of the edge.*

Define $\alpha_i = \max\{\min\{\alpha'_i, \mu\}, AL(i)\}$. The RRank is recursively defined as follows:

$$\begin{aligned}\mathrm{RRank}(v_i) = {} & T_p(v_i, \alpha_i)\big(1 + \beta(1 - \bar{R}_i)^{\alpha_i}\big) \\ & + \max_{v_j \in \mathbf{succ}(v_i)} \big\{\bar{t}_c(e_{i,j}) + \mathrm{RRank}(v_j)\big\}.\end{aligned}$$

where

$$\begin{aligned}\bar{R}_i = {} & (\mathbb{P}\{\text{no failure occurs}\} \\ & + \sum_{k=1}^{\infty} \mathbb{P}\{\text{k recoverable failures occurring,} \\ & v_i \text{ is completed}\}) \\ & \times \prod_{v_k \in \mathbf{pred}(v_i)} \left(1 - \left(1 - \bar{R}[E_{e_{k,i}}]\right)^{\alpha_k}\right) \\ = {} & \bar{R}[E_{v_i}] \sum_{k=0}^{\infty} \big((1 - \bar{R}[E_{v_i}])\gamma\big)^k \\ & \times \prod_{v_k \in \mathbf{pred}(v_i)} \left(1 - \left(1 - \bar{R}[E_{e_{k,i}}]\right)^{\alpha_k}\right) \\ = {} & \frac{\bar{R}[E_{v_i}]}{1 - \gamma + \gamma \bar{R}[E_{v_i}]} \prod_{v_k \in \mathbf{pred}(v_i)} \left(1 - \left(1 - \bar{R}[E_{e_{k,i}}]\right)^{\alpha_k}\right),\end{aligned}$$

β is the weight of the reliability overhead and $\gamma = \sum_{k=1}^{m} \gamma_k / m$ is the mean probability that a failure on processor is recoverable.

4 Makespan Problem

A schedule of a DAG is specified by a starting time and a set of processors that are allocated to every task. Let $\mathrm{EST}(v_i, p_k)$ be the earliest start time of task $v_i \in V$ on a processor $p_k \in P$. The earliest finish time of task v_i on processor p_k is defined as follows.

Definition 3. *A task v_i on a processor p_k with replications of the task on a set of processors S_i that are to be active in the interval from $EST(v_i, p_k)$ to the earliest computation finish time is given by*

$$\begin{aligned}&\mathrm{EFT}(v_i, p_k, S_i) \\ &\quad = \mathrm{EST}(v_i, p_k) + t_p(v_i, k) + \frac{F}{1 - F}\left(\frac{t_p(v_i, k)}{2} + \theta_{p_k}\right)\end{aligned}$$

where $F = \prod_{r \in S_i \cup \{p_k\}} F[E_{v_i, p_r}]$.

The earliest finish time of the communication that is associated with edge $e_{i,j}$ is defined as follows.

Definition 4. *The earliest communication finish time of edge $e_{i,j}$ from $p_k \in S_i$ to p_x is expressed as follows*

$$\begin{aligned}\mathrm{EFT}(e_{i,j}, S_i, p_x) = \frac{1}{|S_i|} \sum_{k \in S_i} & (\mathrm{EFT}(v_i, p_k, S_i) \\ & + t_c(e_{i,j}, l_{k,x})\big).\end{aligned}$$

A schedule is feasible if only one task is being executed by a processor at any time and if the precedence constraints $\mathrm{EFT}(e_{i,j}, S_i, p_x) \leq \mathrm{EST}(v_j, p_x)$ are satisfied for all $v_i \in \mathbf{pred}(v_j)$. Therefore, the following two conditions [26] must be satisfied for all tasks in the DAG.

Condition 1 (The processor constraint). *For any task $v_i, v_j \in V$ that is performed by a processor $p \in P$, $\mathrm{EFT}(v_i, p, S_i) \leq \mathrm{EST}(v_j, p)$ or $\mathrm{EFT}(v_j, p, S_j) \leq \mathrm{EST}(v_i, p)$.*

Condition 2 (The precedence constraint). *For any edge $e_{i,j} \in E$, $v_i, v_j \in V$, $\mathrm{EST}(v_j, p_x) \geq \mathrm{EFT}(e_{i,j}, S_i, p_x)$.*

From Condition 2, the earliest start time of task $v_j \in V$ on a processor $p_x \in P$ is constrained by the entering edges of v_j, and is called the **d**ata **r**eady **t**ime (DRT).

$$\mathrm{DRT}(v_j, p_x) = \max_{e_{i,j} \in E, v_i \in \mathbf{pred}(v_j)} \big\{\mathrm{EFT}(e_{i,j}, S_i, p_x)\big\}.$$

From Conditions 1 and 2, the earliest start time of task v_j on a processor $p_x \in P$ is expressed as follows

$$\mathrm{EST}(v_j, p_x) = \max\big\{\mathrm{DRT}(v_j, p_x), \mathrm{Available}(v_j, p_x)\big\},$$

where $\mathrm{Available}(v_j, p_x)$ is the time available for task v_j on processor p_x.

4.1 The Proposed Algorithm

This section presents a heterogeneous allotment-aware scheduling algorithm, heterogeneous allotment-aware scheduling algorithm (HAAS), in a heterogeneous distributed system. The classic scheduling algorithm with duplication replicates a predecessor of the task to reduce the earliest finish time of that task. A replication method is used herein to reduce the expected processing time. The algorithm utilizes a linear program to obtain an allotment α'. A novel allotment α that is obtained from the resulting allotment α', the allotment parameter μ and the lower bound of allotment $AL(i)$. Let

$$L(v_j, p_k, S_j) = \mathrm{EFT}(v_j, p_k, S_j) + \beta(1-R_j)t_p(v_j,k) + \frac{\beta(1-R_j)F}{1-F}\left(\frac{t_p(v_j,k)}{2} + \theta_{p_k}\right)$$

be the sum of the task reliability overhead and the earliest finish time of task v_j on processor p_k, where

$$R_j = \frac{R[E_{v_j}]}{1-\gamma_k+\gamma_k R[E_{v_j}]} \times \prod_{v_i \in \mathbf{pred}(v_j)} \left(1 - \prod_{p_x \in S_i} F[E_{e_{i,j},p_x,p_k}]\right)$$

and $F = \prod_{r \in S_j \cup \{p_k\}} F[E_{v_j,p_r}]$.

In a previous work [5], malleable tasks were scheduled on a homogenous computing system. All processors that execute a malleable task must start and finish simultaneously. Since a heterogeneous computing system is considered herein, a strategy for choosing a favorable processor is required (*Algorithm 2:* lines 10-12.) Since the replicated tasks do not have to be started simultaneously, the solution α is an upper bound on the number of allotted processors. The algorithm replicates a task until no replication of the task can improve the makespan, or the number of tasks is α (*Algorithm 2:* lines 8, 13-17.)

Algorithm 2. The Allotment-aware Scheduling

1: Normalize the heterogeneous system into a homogenous system.
2: Compute allotment parameter α' by the linear program and the rounding procedure.
3: Compute $\alpha_i = \max\{\min\{\alpha'_i, \mu\}, AL(i)\}$ for $v_i \in V$.
4: Compute RRank for all tasks by traversing graph frome the exit task.
5: Sort the tasks in a scheduling list by non-increasing order of RRank
6: **while** the scheduling list is not empty **do**
7: Remove the first task v_i from the scheduling list and set $S_i = \emptyset$ and $CT = \infty$.
8: **while** $\alpha_i > |S_i|$ **do**
9: Set $L_{min} = \infty$ and $p_c = null$
10: **for** each $p_k \in P \setminus S_i$ **do**
11: **if** $L_{min} > L(v_i, p_k, S_i)$ and $CT > \mathrm{EFT}(v_i, p_k, S_i)$ **then**
12: Set $p_c = p_k$ and $L_{min} = L(v_i, p_k, S_i)$
13: **if** $p_c \neq null$ **then**
14: $S_i = S_i \cup \{p_c\}$
15: $CT = \max_{p_x \in S_i}\{\mathrm{EFT}(v_i, p_x, S_i)\}$
16: **else**
17: No replication of task can improve the makespan and breaks the while-loop.

5 Experimental Results

This section compares the performance of the HAAS algorithm to that of three well-known scheduling algorithms—the HEFT [30], HRDS [29] and GPFA [20] algorithms. The cost function of HEFT is the earliest finish time of a task *without considering the reliability* for a processor. HEFT uses the cost function to choose a favorable processor for a task which is with the minimum earliest finish time. The objective function of GPFA is chosen as the bi-objective aggregation-based heuristic [20]. The HRDS algorithm uses a ready task set Q from a scheduling list and chooses a task in Q to schedule greedily. The latter method improves the performance of all algorithms, but influences the comparison thereof. To avoid this occurrence, $|Q| = 1$ is set.

In testing the algorithms, two sets of graphs, which correspond to randomly generated application graphs and parallel numerical application graphs for real-world problems, are considered. To simulate the recovery of resources, the following random procedure (*Algorithm 3*) is used to determine the execution time of a task. The failure rates of processors and links are assumed to be uniformly distributed between 1×10^{-3} and 1×10^{-2} failures per unit time [30].

Algorithm 3. The Expected Time Procedure

1: Compute the reliability probability $R = e^{-\lambda t_p(v,p)}$ for the task v on the processor p and set $t = t_p(v,p)$.
2: Get a number $n \in [0,1]$ uniformly at random.
3: **while** $n > R$ **do**
4: Get a number $p \in [0,1]$ uniformly at random.
5: $t = t + p \times t_p(v,p) + \theta$
6: Get a number $n \in [0,1]$ uniformly at random.

5.1 Comparison Metrics

The algorithms are compared using the following metrics.

- *Schedule length ratio* [30]. The main performance measure of a scheduling algorithm on a graph is the schedule length (*makespan*) of its output schedule. Since a large set of task graphs with different properties is used, it is necessary to normalize the schedule length to a lower bound, which is called the Schedule length ratio. The SLR value of an algorithm on a graph is defined by

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{min}} \min_{p_j \in P} t_c(v_i, p_j)}.$$

  CP_{min} is the critical path of the DAG. The denominator is the time of critical path that executes on the fastest processor. The SLR of a graph (using any algorithm) cannot be less than one since the denominator is the lower bound. The task-scheduling algorithm that gives the lowest SLR of a graph is the best algorithm with respect to performance. The mean SLR values over several task graphs are used in our simulation experiments.
- *Speedup*. The speedup value for a given graph is computed by dividing the expected sequential execution time by the parallel execution time,

$$Speedup = \frac{\min_{p_j \in P}\left\{\sum_{v_i \in V} T(v_i, p_j)\right\}}{makespan},$$

  where

$$T(v_i, p_j) = t_p(v_i, p_j) + \frac{1 - e^{-\lambda_j t_p(v_i,p_j)}}{e^{-\lambda_j t_p(v_i,p_j)}}\left(\frac{t_p(v_i,p_j)}{2} + \theta_{p_j}\right).$$

- *Reliability probability*. In the section 2.1, the reliability probability of tasks is independent on those of edges. However, to evaluate the application reliability probability, the reliability probability of edges are influenced those of tasks. To improve discrimination between the definition of Section 2.1 and the evaluation of application reliability probability, let $\mathbb{P}[E_{v_i,p_x}]$ denote the application reliability probability of v_i executed on processor p_x. Since Strategy II aborts other replication of tasks, the reliability probability is dependent on which processor has completed task. Let C_i denote a set of processors which has completed task v_i. The application reliability probability of v_j is

$$\begin{aligned}\mathbb{P}[E_{v_j,p_k}] &= (\mathbb{P}\{\text{no failure occurs}\} \\ &\quad + \sum_{k=1}^{\infty} \mathbb{P}\{k \text{ recoverable failures} \\ &\quad . \text{ occurring}, v_j \text{ is completed}\}) \\ &\quad \times \prod_{v_i \in \mathbf{pred}(v_j)} Re(e_{i,j}, p_k) \\ &= R[E_{v_j,p_k}] \sum_{i=0}^{\infty} \Big(\Big(1 - R[E_{v_j,p_k}]\Big)\gamma_k \Big)^i \\ &\quad \times \prod_{v_i \in \mathbf{pred}(v_j)} Re(e_{i,j}, p_k) \\ &= \frac{R[E_{v_j,p_k}]}{1 - \gamma_k + \gamma_k R[E_{v_j,p_k}]} \\ &\quad \times \prod_{v_i \in \mathbf{pred}(v_j)} Re(e_{i,j}, p_k)\end{aligned}$$

where

$$Re(e_{i,j}, p_k) = 1 - \prod_{p_x \in C_i} \Big(1 - R[E_{e_{i,j},p_x,p_k}]\Big).$$

The application reliability probability is defined as follows

$$\mathbb{P}[G] = \prod_{v_i \in V} \left(1 - \prod_{p_x \in S_i} \left(1 - \mathbb{P}[E_{v_i,p_x}]\right)\right).$$

5.2 Randomly Generated Application Graphs

This section first considers a set of randomly generated application graphs that is formed with a set of fundamental characteristics.

- *DAG size*, n. The number of tasks in the application DAG.
- *Communication to computation cost ratio, CCR*. The mean communication cost divided by the mean computation cost of the application DAG.
- *Out degree*, d. Out degree of a task node.
- *Parallelism factor*, (α) [6]. The number of levels of the application DAG is calculated by randomly generating a number, using a uniform distribution with a mean value of $\frac{\sqrt{n}}{\alpha}$, and then rounding it up to the nearest integer. Also, the width of each level is calculated by randomly generating a number using a uniform distribution with a mean value of $\alpha \times \sqrt{n}$ and the rounding it up to the nearest integer [30]. A dense graph can be generated by selecting $\alpha \gg 1.0$ and a low α value leads to a DAG with a low parallelism degree [3], [27].
- *Computational cost heterogeneity factor*, (h) [6]. A high h value indicates high variance of the computational costs of a task, with respect to the processors in the system, and *vice versa*. If the heterogeneity factor is set to 0, the computational cost of a task is the same for all processors. The mean computational cost of a task $\overline{w(v_i)}$ is randomly generated using a uniform distribution generator with a mean value of W. The value of W does not affect the performance results of the scheduling algorithms. If there are m processors in a heterogeneous distributed systems, the computational cost of a task v_i for each node is set by randomly selecting m computational cost values of v_i from the range $[\overline{w(v_i)} \times (1 - \frac{h}{2}), \overline{w(v_i)} \times (1 + \frac{h}{2})]$.

Let $level_i$ be the level of task v_i and $Lv(v_i) = \{v_j \in V | level_i < level_j\}$ be a set of tasks that have a higher level than v_i. In the construction of an acyclic DAG, only a subset of $Lv(v_i)$ can be chosen as successors of each task v_i. For the task v_i, a set of successors is randomly chosen with a probability of $\frac{d}{|Lv(v_i)|}$. Let W be the mean weight of a task. Task weights are generated randomly with a uniform distribution in $[1, 2W - 1]$. Edge weights are generated with a uniform distribution $[1, 2CCR \times W - 1]$. Therefore, the various types of applications that are related to CCR and W can be constructed. The frequency of the processor (the number of unit times per operation) and the frequency of the link (the number of unit times per data) are randomly determined with a uniform distribution in $[1 - \frac{h}{2}, 1 + \frac{h}{2}]$. Therefore, the expected processing time of a task v_i at each node is in the range $[\overline{w(v_i)} \times (1 - \frac{h}{2}), \overline{w(v_i)} \times (1 + \frac{h}{2})]$. This experiment assumes that the recovery time depends on the capacity of the processor. The recovery times $\theta_{p_i} = 1/w(p_i)$ for all processor $p_i \in P$. The probability γ_i, that a failure on processor p_i is recoverable, is obtained randomly and uniformly distributed in $[0.5, 0.9]$ for all processors $p_i \in P$.

5.3 Random Application Performance Results

In this section, each data point is the mean of data obtained in 1,000 experiments. Let HAAS1 be the proposed algorithm that implements Strategy I; HAAS2 be the proposed algorithm that implements Strategy II and HAAS3 be the proposed algorithm with $\alpha_i = 1$ for all $v_i \in V$.

Figs. 1, 2, and 3 present numerical results concerning the SLR, reliability and speedup for various CCR values. Graphs are generated from $W = 30, h = 0.5, \alpha = 1, d = 3$ and the number of tasks between 40 and 200, in steps of 40. The weight of the reliability overhead β is zero. Figs. 1, 2, and 3 reveal that HAAS significantly outperforms HEFT, HRDS and GPFA in terms of the SLR, reliability and speedup. The SLR, reliability and speedup that are generated using the algorithms with computation-intensive applications (CCR = 0.1) indicate that HAAS2 outperforms other algorithms except HAAS3 in case of a large application with $n \geq 160$.

For communication-intensive applications (CCR = 3), HAAS3 significantly outperforms other algorithms in terms

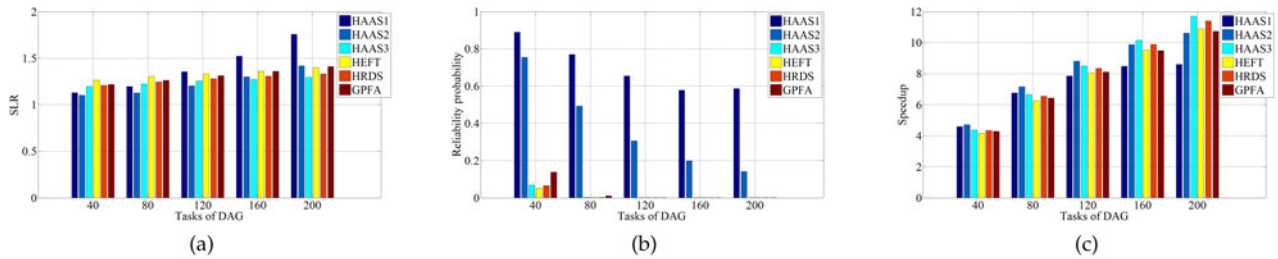

Fig. 1. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for CCR = 0.1.

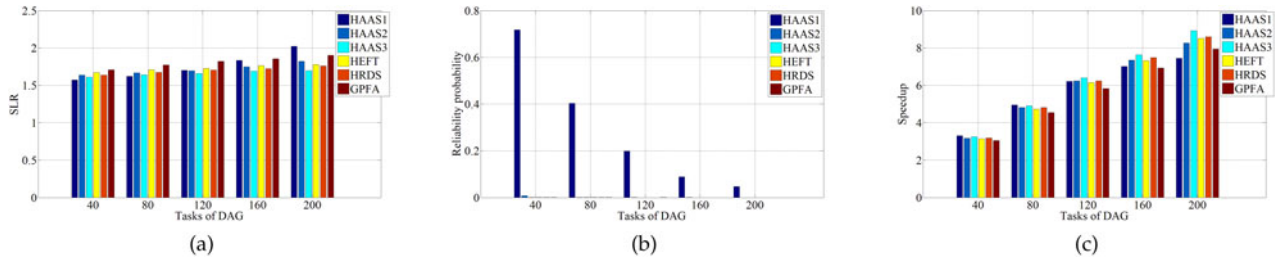

Fig. 2. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for CCR = 1.

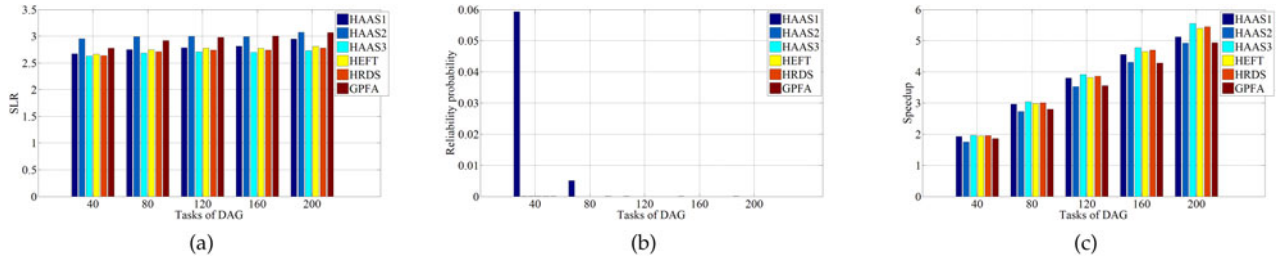

Fig. 3. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for CCR = 3.

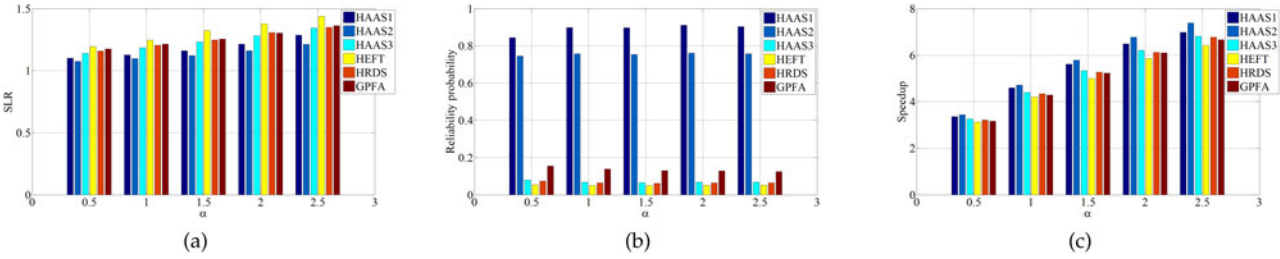

Fig. 4. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for CCR = 0.1 and DAG size = 40.

of the SLR and the speedup, mainly because the communication dominates the other terms in the performance. Since the proposed algorithm solves the allotment problem without communication cost, the replication of a task cannot improve the makespan. In Strategy I, no replication of a task can be stopped, then it has many replication of edge to reduce the communication time. Therefore, Strategy I outperforms Strategy II.

In the computation-intensive applications (CCR ≤ 1), the slope of speedup in HAAS3 exceeds the slope of speedup in HAAS1 and HAAS2 because the processors are too few. The following brief conclusion can be drawn. HAAS3 performs better when the ratio n/m or CCR is larger. Strategy I is preferred for communication-intensive applications and Strategy II is preferred for computation-intensive applications. The proposed algorithm using Strategy I outperforms the other algorithms in terms of reliability.

A second set of experiments on the graph structure is considered. The parallelism factor α is set from 0.5 to 2.5. Fig. 4 reveals that HAAS significantly outperforms the other algorithms in terms of the SLR, reliability and speedup. Since highly parallel applications can be parallelized effectively in a multiprocessor system, the performance increases significantly as the parallelism factor α increases.

A third set of experiments are performed on the heterogeneity factors h that between zero to one. Fig. 5 reveals that HAAS2 outperforms HEFT, HRDS and GPFA in terms of the SLR, reliability and speedup when $h \leq 0.75$. When $h = 1$, HAAS3 outperforms HEFT, HRDS and GPFA in terms of the SLR, reliability and speedup. HAAS1 and HAAS2 perform better at a lower heterogeneity factor.

A fourth set of experiments are performed on the weight of reliability overhead β. A trade-off between the performance and reliability is made using β. Fig. 6 reveals that HAAS with

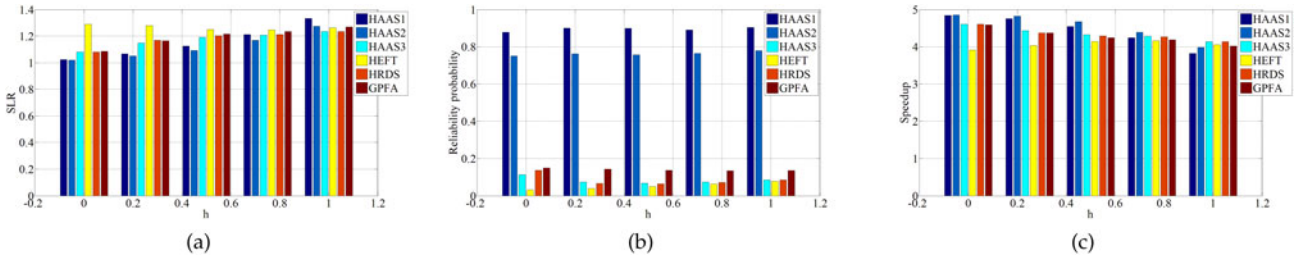

Fig. 5. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for various heterogeneity factors h.

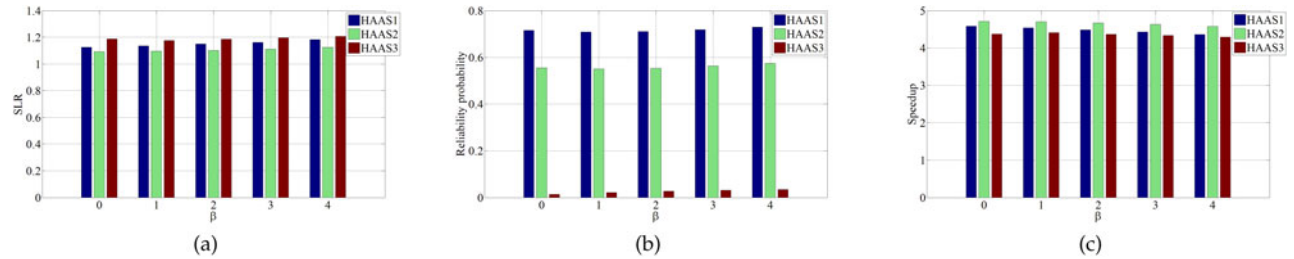

Fig. 6. SLR, Reliability probability and Speedup of HAAS for various weights of reliability overhead β where $\gamma_i \in [0.1, 0.5]$.

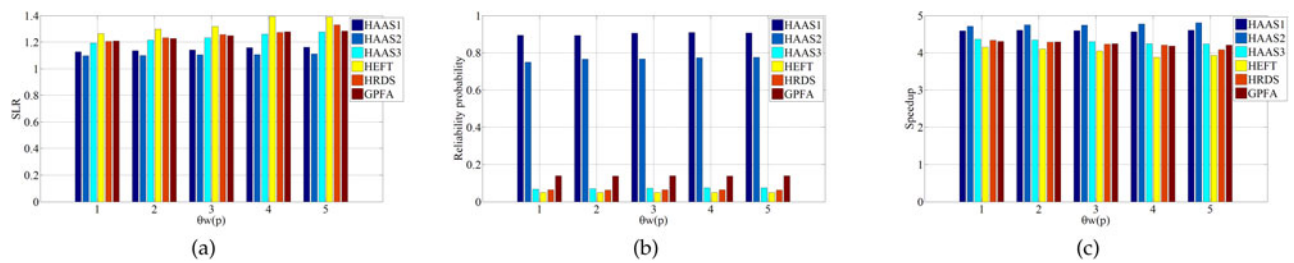

Fig. 7. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for various recovering time θ.

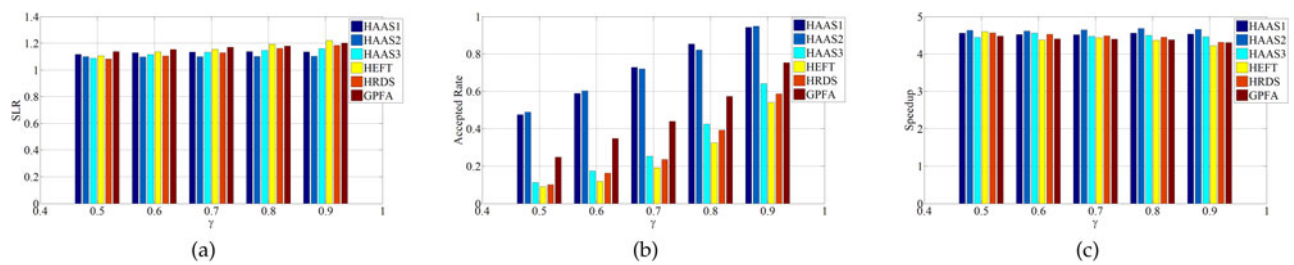

Fig. 8. SLR, Accepted rate and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for various γ.

a higher β has a greater reliability than HAAS with a lower β. However, the performance becomes worse as β increases.

A fifth set of experiments are performed on the recovery time. Fig. 7 reveals that SLR increases with the recovery time. Recovery time affects the speedup of HEFT, HRDS and GPFA but not that of HAAS.

A sixth set of experiments are performed on the probability γ. In these experiments, a processor may fail and cannot recover. Therefore, scheduling may not be completed. This experiment concerns another important metric that is the accepted rate of scheduling (the number of successful schedules/the total number of schedules). Fig. 8b reveals that the accepted rate increases with γ. Moreover, Fig. 8 reveals that HAAS2 outperforms HEFT, HRDS and GPFA in terms of the SLR, accepted rate and speedup.

A seventh set of experiments are performed on the size of Q. Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2015.2403861 available online, presents the pseudocode of the local search method associated with the proposed algorithm. Fig. 9 reveals that the local search method cannot greatly improve the performance of algorithms. HAAS significantly outperforms the other three in terms of the SLR, the reliability and the speedup.

5.4 Regular Application Graphs

This section evaluates the performance of the scheduling algorithms in three real-world parallel problems: Gaussian elimination, fast Fourier transform and a molecular dynamics code [6].

5.4.1 *Gaussian Elimination*

In the experiments on the Gaussian elimination application, CCR = 0.1 and the computation cost heterogeneity factor h was fixed. Since the structure of the application graph is

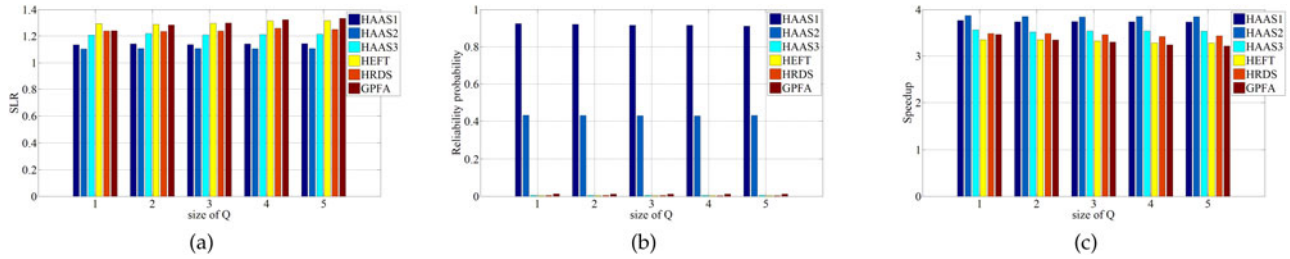

Fig. 9. SLR, Reliability probabilityand Speedup of HAAS, HRDS, HEFT and GPFA algorithms for various size of Q.

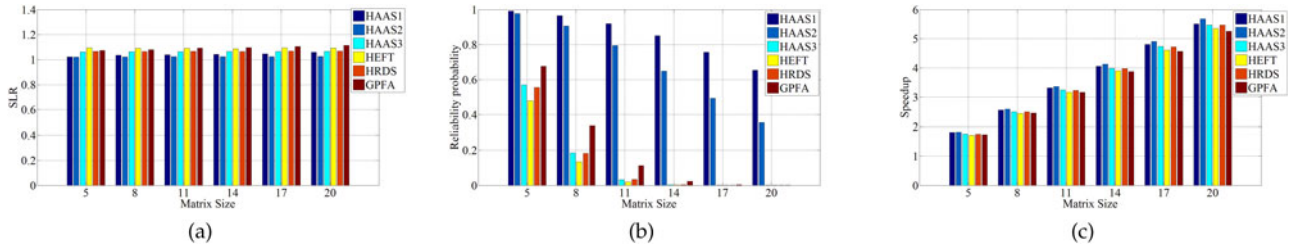

Fig. 10. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for the Gaussian elimination graph.

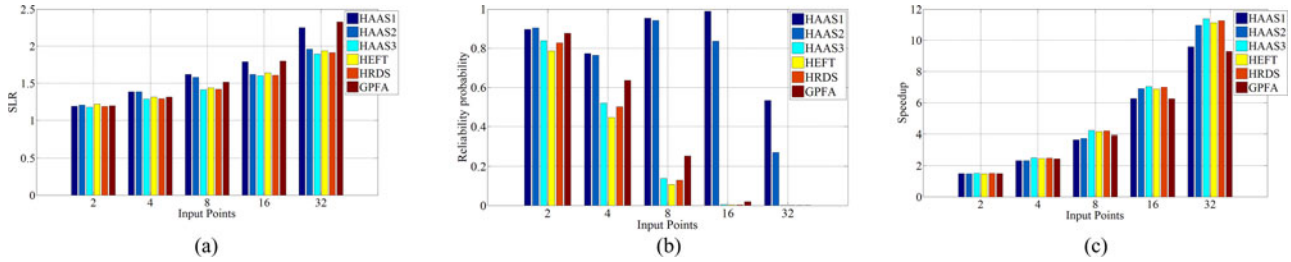

Fig. 11. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for the FFT graph.

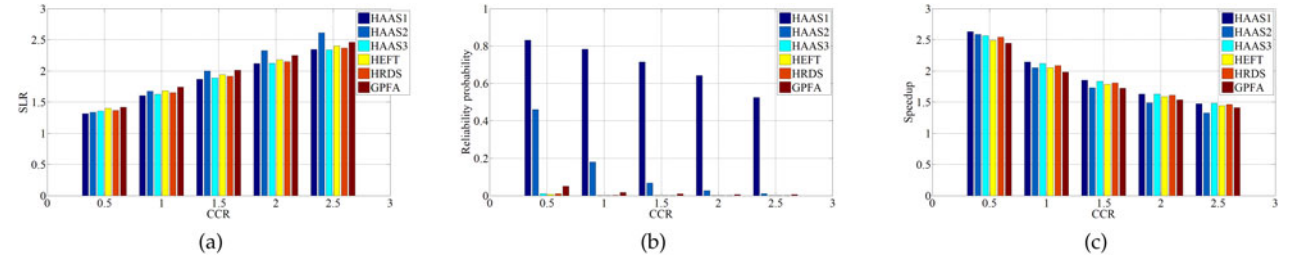

Fig. 12. SLR, Reliability probability and Speedup of HAAS, HRDS, HEFT and GPFA algorithms for the task graph of a molecular dynamics code.

known, the other parameters related to the graph structure are not required. A new parameter, matrix size (Z), is used instead of DAG size n (the number of tasks in the DAG graph). The total number of tasks in a Gaussian elimination graph is $\frac{Z^2+Z-2}{2}$ [30].

Fig. 10 presents the SLR, reliability and speedup of algorithms with various matrix sizes from 5 to 20 in increments of three, with the number of processors set to 32. As the matrix size increases, more tasks are not on the critical path, improving the performance of each algorithm. The HAAS2 significantly outperforms the HRDS, HEFT and GPFA algorithms in terms of the SLR, reliability and speedup.

5.4.2 Fast Fourier Transformation (FFT)

The recursive, one-dimensional FFT algorithm and its task graph can be found elsewhere [27], [30]. The FFT algorithm with an input vector of size Z has $2Z-1$ recursive call tasks and $Z \log Z$ butterfly operation tasks. In the experiments on FFT applications, the CCR and the computational cost heterogeneity factor h were fixed. The numbers of data points in FFT are powers of two from 2 to 32.

Fig. 11 reveals that HAAS3 outperforms HEFT, HRDS and GPFA in terms of the SLR, reliability and speedup. The slope of speedup in HAAS3 exceeds the slop of speedup in HAAS1 and HAAS2.

5.4.3 Molecular Dynamics Code

The task graph of a modified molecular dynamic code can be found elsewhere [30]. Since the structure of the application graph is known, only the values of CCR and the computational cost heterogeneity factor h are used to simulate. The CCR is increased from 0.5 to 2.5 in steps of 0.5.

Fig. 12 reveals that HAAS outperforms the other algorithms in terms of the SLR, reliability and speedup. As CCR increases, the performance of the algorithms becomes worse. The HAAS1 outperforms HRDS, HEFT and GPFA in terms of the SLR, reliability and speedup. When CCR is at least two, HAAS3 outperforms the other algorithms in terms of the SLR and speedup.

6 Concluding Remarks

This work considers the expected makespan to schedule tasks in heterogeneous distributed systems. A heterogeneous allotment-aware scheduling algorithm is proposed. The algorithm is a two-phase algorithm. In the first phase, a favorable allotment is determined using a linear program formulation and a rounding procedure. In the second phase, a scheduling method that is based on the expected execution time and the communication time is implemented.

Two strategies I and II are considered. In Strategy I, no replication of a task can be stopped. In Strategy II, once a replication of a task is completed, the other replications of a task are immediately aborted. Strategy I is preferred for communication-intensive applications and Strategy II is preferred for computation-intensive applications. According to an experimental study in which a set of randomly generated application graphs and application graphs of three real-world problems (Gaussian elimination, FFT, and a molecular dynamics code) were used, the HAAS algorithm significantly outperformed the other algorithms in terms of the SLR, reliability and speedup.

Acknowledgments

C.-Y. Chen is the corresponding author.

References

[1] I. Ahmad and Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 9, pp. 872–892, Sep. 1998.

[2] I. Assayad, A. Girault, and H. Kalla, "A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints," in *Proc. Int. Conf. Dependable Syst. Netw.*, Jun. 2004, pp. 347–356.

[3] S. Bansal, P. Kumar, and K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533–544, Jun. 2003.

[4] M. S. Bouguerra, D. Kondo, F. Mendonca, and D. Trystram, "Fault-tolerant scheduling on parallel systems with non-memoryless failure distributions," *J. Parallel Distrib. Comput.*, vol. 74, no. 5, pp. 2411–2422, 2014.

[5] C.-Y. Chen and C.-P. Chu, "A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1479–1488, Aug. 2013.

[6] M. I. Daoud and N. Kharma, "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 68, no. 4, pp. 399–409, 2008.

[7] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 308–323, Mar. 2002.

[8] A. Dogan and F. Ozguner, "Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems," *Comput. J.*, vol. 48, no. 3, pp. 300–314, 2005.

[9] H. El-Rewini and T. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *J. Parallel Distrib. Comput.*, vol. 9, no. 2, pp. 138–153, 1990.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA; Freeman, 1979.

[11] A. Girault and H. Kalla, "A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 4, pp. 241–254, Oct. 2009.

[12] A. Girault, Érik Saule, and D. Trystram, "Reliability versus performance for critical applications," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 326–336, 2009.

[13] S. Guo, H.-Z. Huang, Z. Wang, and M. Xie, "Grid service reliability modeling and optimal task scheduling considering fault recovery," *IEEE Trans. Rel.*, vol. 60, no. 1, pp. 263–274, Mar. 2011.

[14] M. Hakem and F. Butelle, "A bi-objective algorithm for scheduling parallel applications on heterogeneous systems subject to failures," in *Proc. RenPar*, 2006, pp. 25–35.

[15] K. Hashimoto, T. Tsuchiya, and T. Kikuno, "A new approach to realizing fault-tolerant multiprocessor scheduling by exploiting implicit redundancy," in *Proc. 27th Annu. Int. Symp. Fault-Tolerant Comput. Digest Papers.*, Jun. 1997, pp. 174–183.

[16] K. Hashimoto, T. Tsuchiya, and T. Kikuno, "Effective scheduling of duplicated tasks for fault tolerance in multiprocessor systems," *IEICE Trans. Inform. Syst.*, vol. 85, no. 3, pp. 525–534, 2002.

[17] Y. He, Z. Shao, B. Xiao, Q. Zhuge, and E. Sha, "Reliability driven task scheduling for heterogeneous systems," in *Proc. 15th IASTED Int. Conf. Parallel Distrib. Comput. Syst.*, 2003, vol. 1, pp. 465–470.

[18] M. A. Iverson, F. Özgüner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in *Proc. 4TH Heterogeneous Comput. Workshop*, 1995, pp. 93–100.

[19] K. Jansen and H. Zhang, "Scheduling malleable tasks with precedence constraints," *J. Comput. Syst. Sci.*, vol. 78, no. 1, pp. 245–259, 2011.

[20] E. Jeannot, E. Saule, and D. Trystram, "Optimizing performance and reliability on heterogeneous parallel systems: Approximation algorithms and heuristics," *J. Parallel Distrib. Comput.*, vol. 72, no. 2, pp. 268–280, 2012.

[21] J. Plank and W. Elwasif, "Experimental assessment of workstation failures and their impact on checkpointing systems," in *Proc. 28th Annu. Int. Symp. Fault-Tolerant Comput.*, Jun. 1998, pp. 48–57.

[22] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proc. 5th IEEE/ACM Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2007, pp. 233–238.

[23] X. Qin and H. Jiang, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems," *Parallel Comput.*, vol. 32, no. 5, pp. 331–356, 2006.

[24] S. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Rel.*, vol. 38, no. 1, pp. 16–27, Apr. 1989.

[25] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[26] O. Sinnen, L. Sousa, and F.-E. Sandnes, "Toward a realistic task scheduling model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 3, pp. 263–275, Mar. 2006.

[27] X. Tang, K. Li, R. Li, and B. Veeravalli, "Reliability-aware scheduling strategy for heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 941–952, 2010.

[28] X. Tang, K. Li, and D. Padua, "Communication contention in APN list scheduling algorithm," *Sci. China Series F: Inform. Sci.*, vol. 52, no. 1, pp. 59–69, 2009.

[29] X. Tang, K. Li, M. Qiu, and E. H.-M. Sha, "A hierarchical reliability-driven scheduling algorithm in grid systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 4, pp. 525–535, 2012.

[30] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[31] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 951–967, Sep. 1994.

Chi-Yeh Chen received the BS degree in communication engineering from Da-Yeh University, Changhua, Taiwan, ROC, in 2001, and the MS degree in computer science and information and engineering from National Cheng Kung University, Tainan, Taiwan, ROC, in 2005, and the PhD degree in computer science and information and engineering from National Cheng Kung University, Tainan, Taiwan, ROC, in 2012. He is currently an assistant researcher in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include scheduling problems, approximation algorithms, parallel algorithm and load distribution.