J Grid Computing (2017) 15:23–39
DOI 10.1007/s10723-016-9386-7

A Reliability-aware Task Scheduling Algorithm Based on Replication on Heterogeneous Computing Systems

Shuli Wang · Kenli Li · Jing Mei · Guoqing Xiao · Keqin Li

Received: 12 April 2014 / Accepted: 1 November 2016 / Published online: 30 November 2016

Abstract Over the past several years, a heterogeneous computing (HC) system has become more competitive as a commercial computing platform than a homogeneous system. With the growing scale of HC systems, network failures become inevitable. To achieve high performance, communication reliability should be considered while designing reliability-aware task scheduling algorithms. In this paper, we propose a new algorithm called RMSR (Replication-based scheduling for Maximizing System Reliability), which incorporates task communication into system reliability. To maximize communication reliability, an improved algorithm which searches all optimal reliability communication paths for current tasks is proposed. During the task replication phase, the task reliability threshold is determined by users and each task has dynamic replicas. Our comparative studies for both randomly generated graphs and application graphs of real-world problems show that our RMSR algorithm outperforms existing scheduling algorithms in terms of system reliability. For randomly generated graphs, several factors affecting the performance are analyzed in the paper. For an application graph of a real-world problem with a fixed DAG, the system reliability of the RMSR algorithm is at most influenced by one factor.

The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the International Science & Technology Cooperation Program of China (Grant No. 2015DFA11240), the National Key R&D Program of China (Grant No. 2016YFB0201402), the Hunan Provincial Innovation Foundation For Postgraduate (Grant No. CX2016B066), and the Outstanding Graduate Student Innovation Fund Program of Collaborative Innovation Center of High Performance Computing.

S. Wang · K. Li (✉) · J. Mei · G. Xiao · K. Li
College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China
e-mail: lkl@hnu.edu.cn

S. Wang
e-mail: rosa_wsl@hnu.edu.cn

J. Mei
e-mail: jingmei1988@163.com

G. Xiao
e-mail: xiaoguoqing@hnu.edu.cn

K. Li
e-mail: lik@newpaltz.edu

K. Li
National Supercomputing Center in Changsha, Changsha, Hunan 410082, China

K. Li
Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

S. Wang
Hunan Zhongyi Communication Technology Engineering Co., Ltd., Changsha, Hunan, 410003, China

J. Mei
College of Mathematics and Computer Science, Hunan Normal University, Changsha, Hunan, 410006, China

Keywords Directed acyclic graph · Heterogeneous computing systems · Reliability-aware scheduling · Replication-based algorithm

1 Introduction

In the past decade, more and more attention has been focused on the problem of scheduling applications on heterogeneous computing (HC) systems. For high performance computing and information processing, HC systems have become a popular and powerful commercial platform over the past several years. With the growing scale of HC systems, machine and network failures become inevitable. Ensuring high reliability of HC systems becomes an important issue.

The directed acyclic graph (DAG) is a traditional representation of a parallel application. In a DAG, nodes represent application tasks and the directed edges represent inter-task dependencies, such as precedence constraints. Task scheduling is to assign tasks of an application to processors, so that precedence requirements are satisfied and the minimum makespan can be achieved [15, 21]. However, it is in general NP-hard [17, 33]. Therefore, heuristics can be used to acquire sub-optimal schedules. The general task scheduling algorithms can be classified into several categories, such as list scheduling algorithms, cluster algorithms, duplication-based algorithms, and so on.

List scheduling is a very popular method [22]. The basic process of list scheduling is to compute priorities for the tasks of a DAG and rank all the tasks in non-increasing order of priorities. Recently, a few variants of the list scheduling algorithm have been proposed to deal with HC systems, such as predict earlist finish time (PEFT) algorithm [1], dynamic-level scheduling (DLS) algorithm [24], mapping heuristic (MH) [10], levelized-min time (LMT) algorithm [13], and heterogeneous earliest-finish-time (HEFT) algorithm [3, 16, 31]. The HEFT algorithm significantly outperforms the DLS, MH, and LMT algorithms in terms of average schedule length ratio, speedup, etc. [16, 31].

Unfortunately, these algorithms do not consider the probability of failure of the machines and relevant network resources. Furthermore, most of these algorithms are based on a very simple system model, which does not accurately reflect real parallel systems. The main assumptions are summarized by the following three points:

- A dedicated subsystem for inter-processor communication.
- A fully connected communication network.
- A network that never fails during communication.

However, in a real world, the failure of processors and networks is inevitable. If failure occurs, it may result in restarting of an application, thereby increasing the execution time of the application. In order to solve this problem, some scheduling algorithms take reliability into account. Dogan and Özgüner proposed three reliability cost functions that were incorporated into making dynamic level (DL) and presented a reliability dynamic level scheduling RDLS algorithm [7, 8], which minimizes not only the execution time but also the probability of failure of an application. Kartik et al. [14] proposed an algorithm that can maximize the system reliability using the idea of branch and bound. Zhao et al. [38] proposed an algorithm in order to maximize the overall reliability under given time and energy constraints. Dongarra et al. [9] designed two algorithms that optimize both makespan and reliability.

In this paper, our main objective is to propose a replication-based algorithm which maximizes the system reliability while considering the communication between tasks. By comparing with the RASD algorithm [26], our algorithm can achieve higher system reliability. It is because our replication method guarantees the reliability of each task to be higher than the task reliability threshold γ which is determined by users.

The main contributions of this paper are summarized as follows.

- We propose a replication-based scheduling algorithm which aims at maximizing system reliability. The number of replicas of each task is computed by comparing to task reliability threshold. Moreover, it considers the communication between two precedence constrained tasks in a heterogeneous distributed system.
- An improved algorithm which searches all optimal reliability communication paths for the current tasks is proposed for enhancing the communication reliability.
- Experiments are conducted to verify that the proposed algorithm can achieve high system reliability.

The remainder of this paper is organized as follows. In Section 2, we review related work. In Section 3, we describe related models and definitions, where an example is also given to illustrate our algorithm. In Section 4, we generically analyze the reliability of a communication system and propose an improved algorithm to enhance the communication reliability. In Section 5, we give a detailed description of the RMSR algorithm. The experimental results are presented in Section 6, together with the analysis of these experiments. Section 7 concludes the work of our paper and provides an overview of future research.

2 Related Work

More and more people are paying attention to the improvement of reliability. A reliable scheduling scheme can greatly improve a efficiency of the schedule [20]. As a result, reliability analysis based scheduling algorithms have been addressed by many works [28, 32, 34]. In [28], Tang et al. designed a hierarchical reliability-driven scheduling architecture that includes both a local scheduler which aims to effectively measure task reliability of an application in a grid virtual node and a global scheduler in which they proposed a hierarchical reliability-driven scheduling algorithm based on quantitative evaluation. In [32], Tosun presented an integer linear programming (ILP) based framework that maps a given task set onto a heterogeneous multiprocessor system-on-Chip (HMPSoC) architecture. They employed task duplication to maximize the reliability. One task only has at most one duplicate. In [34], Wang et al. proposed a lookahead genetic algorithm (LAGA) which utilizes the reputation to optimize both the makespan and the reliability of a workflow application.

Redundancy is a popular technique to improve reliability of distributed systems [4, 11, 12, 30, 36]. There are two kinds of redundancy, i.e., software redundancy [4, 30] and hardware redundancy [11, 12]. Hardware redundancy is an expensive approach. Furthermore, the hardware configuration is confirmed. Hence, we prefer to achieve high reliability through replication. For resource replication, there are two main schemes described as follows.

- Active replication scheme: Several processors are scheduled simultaneously and tasks will succeed if at least one processor does not encounter a failure [2, 39, 40]. In [2], Benoit et al. presented the FTSA algorithm which uses ε +1 replicas for each task to guarantee the system reliability. This will lead to large resource redundancy which has an adverse impact for the system performance. In [40], Zhao et al. designed the MaxRe algorithm in which tasks have different numbers of replicas. However the computing system model of MaxRe algorithm is fully connected which is different from our's. Hence, we do not compare our algorithm with MaxRe in the experiment section. In [39], Zhao et al. proposed the deadline, reliability, resources-aware (DRR) scheduling algorithm to use the minimum resources.
- Primary/backup scheme: When a primary processor encounters a failure, a task will be rescheduled on a backup processor [27, 29, 37, 41]. In [41], Zheng et al. identified two cases that may happen when scheduling dependent tasks with the primary-backup approach, independent tasks and dependent tasks, respectively. In [37], Zhang et al. addressed the problem of building a reliable and highly-available grid service by replicating the service on two or more hosts using the primary-buckup approach. In [29], Tao et al. proposed a Markov chain based grid node availability prediction model which can efficiently predict grid nodes availability in the future without adding significant overhead. In [27], Tang et al. built an application reliability analysis model based on Weibull distribution, which can dynamically measure the reliability of task executing on heterogeneous cluster with arbitrary network architectures. Furthermore, to improve system reliability, they duplicates task as if task hazard rate is more than threshold θ. The method we use to improve system reliability is setting a task reliability threshold γ.

However, some of these algorithms do not consider communication between tasks as well as failure of the links in a network. In other words, most of these algorithms are based on a very simple system model described in Section 1.

In [7, 8, 14, 38] mentioned in Section 1, and [9], these algorithms consider both makespan and system reliability based on a realistic model. In [9], Dongarra et al. provided an optimal scheduling algorithm for independent unitary tasks where the

objective is to maximize the reliability subject to makespan minimization. They are able to let the user choose a trade-off between reliability maximization and makespan minimization. In [26], Tang et al. designed a reliability-driven scheduling algorithm (RASD) which could effectively measure system reliability. Even though the RASD algorithm considers both reliability and makespan, the improvement of system reliability that RASD generates is not that much. Obviously, the reliability achieved by these research is limited. The reason is analyzed as follows. We all know that there is inverse relationship between reducing schedule length and improving the system reliability of an application. Even though the algorithms mentioned above consider both schedule length and reliability, the improvement of system reliability is limited. In other words, the system reliability still cannot achieve a high value especially in a failure-prone system. It is quite possible that a large and long-running application may experience a failure during the whole execution time. Therefore, the application restarts. As a result, the actual execution time of the application is increased.

3 Models, Notations, and Definitions

In this section, we introduce two models which are tightly connected to task scheduling. One is an application model which is represented as a DAG graph, and the other is a computing system model which is represented as a topology graph. For the reader's convenience, we summarize the notations and their definitions used in this paper in Table 1.

Table 1 Notations and Definitions

Notations	Definitons
T	A set of weighted tasks
P	A set of heterogeneous processors
E	A set of weighted and directed edges representing communications among tasks in T
γ	Task reliability threshold
$\lambda_{i,j}$	The failure rate of processor p_j when executing task t_i
$\lambda_{l_{i,j}}$	The failure rate of link $l_{i,j}$ from processor p_i to p_j
$s_{i,j}$	The computation capacity of processor p_j when executing task t_i
$f_{i,j}$	The frequency of processor p_j when executing task t_i
$w(t_i)$	The computation cost of task t_i
$e_{i,j}$	The communication edge between tasks t_i and t_j in E
$c_{i,j}$	The communication cost between tasks t_i and t_j
$succ(t_i)$	A set of tasks which are direct successors of task t_i
$pred(t_i)$	A set of tasks which are direct predecessors of task t_i
$proc(t_i)$	A set of processors executing task t_i
$drt(t_i, p_j)$	The data ready time of task t_i when it is executed on processor p_j
$est(t_i, p_j)$	The earliest start time of task t_i on processor p_j
$et(t_i, p_j)$	The execution time of task t_i when it is executed to processor p_j
$\overline{et(t_i)}$	The average computation time of task t_i
$eft(t_i, p_j)$	The earliest finish time of task t_i on processor p_j
$pr(t_i, p_j)$	The present reliability of task t_i which is executed on processor p_j
$fr(t_i)$	The final reliability of task t_i after replication
$lest(e_{i,j}, p_k, p_n, l_i)$	The earliest start time of link l_i for edge $e_{i,j}$ transferred from p_k to p_n
$left(e_{i,j}, p_k, p_n, l_i)$	The earliest finish time of link l_i for edge $e_{i,j}$ transferred from p_k to p_n
$r(t_i, p_j)$	The reliability of processor p_j during the execution of task t_i without considering the predecessors of t_i
$r(e_{i,j}, p_k, p_n, l_i)$	The reliability of link l_i for $e_{i,j}$ from p_k to p_n
$cet(e_{i,j}, p_k, p_n)$	The execution time of communication edge $e_{i,j}$ transferred from p_k to p_n
$r(e_{i,j}, p_k, p_n)$	The reliability of $e_{i,j}$ when the data is transferred from p_k to p_n

3.1 Application Model

A parallel application is usually represented by a directed acyclic graph (DAG) $G_t = \langle T, E, w, c\rangle$, where T, E, w, and c are the set of m task nodes, the set of communication edges, the set of computation costs associated with the task nodes, and the set of communication costs associated with the edges, respectively. The weight $w(t_i)$ is the computation cost of task t_i. In this model, the execution time of task t_i when it is executed on processor p_j is represented by $et(t_i, p_j)$. It can be calculated by the following equation:

$$et(t_i, p_j) = \frac{w(t_i)}{f_{i,j}}, \tag{1}$$

where $f_{i,j}$ denotes the frequency of processor p_j when executing task t_i according to the voltage and frequency scaling technique [38]. A speed $s_{i,j}$ denotes the computation capacity (the amount of computation that can be performed in a unit of time) of processor p_j when executing task t_i. The relationship between them follows the following equation:

$$s_{i,j} \propto f_{i,j}. \tag{2}$$

The average computation time $\overline{et(t_i)}$ of task t_i can be given by the following equation,

$$\overline{et(t_i)} = \frac{1}{n}\sum_{j=1}^{n} et(t_i, p_j), \tag{3}$$

where n is the number of heterogeneous processors in an HC system.

In this model, $pred(t_i)$ denotes the set $\{t_x \in T : e_{x,i} \in E\}$ of all the direct predecessors of task t_i. Similarly, $succ(t_i)$ denotes the set $\{t_x \in T : e_{i,x} \in E\}$ of all the direct successors of task t_i. If a task t satisfies the equation $pred(t) = \emptyset$, then it is called a source task. If a task t satisfies the equation $succ(t) = \emptyset$, then it is called a sink task. In this paper, we allow that there are more than one source task or more than one sink task in a DAG.

The following terms describe a schedule S of a DAG $G_t = \langle T, E, w, c\rangle$ on an HC system. The earliest start time of task t_i executed on processor p_j is denoted by $est(t_i, p_j)$. Similarly, the earliest finish time of task t_i executed on processor p_j is denoted by $eft(t_i, p_j) = est(t_i, p_j) + et(t_i, p_j)$. The set $proc(t)$ denotes the set of all processors to which task t is assigned.

The edge $e_{i,j}$ represents the precedence constraint between tasks t_i and t_j. In other words, task t_j can start the execution only after all the replicas of t_i are finished. Let $c_{i,j}$ be the communication cost associated with edge $e_{i,j}$. The execution of task computations on a processor is sequential. Meanwhile, a computation cannot be divided into several parts. Hence, when scheduling starts, these edges must satisfy the following conditions.

Condition 1 (Precedence Constraint) Let S be a schedule for task graph $G_t = \langle T, E, w, c\rangle$ on a heterogeneous computing system P. For $t_i, t_j \in T, e_{i,j} \in E$, $p_x, p_y \in P$:

$$est(t_j, p_y) \geq \max_{p_x \in proc(t_i)} \{eft(t_i, p_x)\}, \tag{4}$$

that is, p_y cannot start until all replicas of p_x are completed.

Condition 2 (Exclusive Processor Allocation) Let S be a schedule for task graph $G_t = \langle T, E, w, c\rangle$ on a heterogeneous computing system P. For any two tasks t_i and $t_j \in T$, a processor $q \in P$:

$$q \in proc(t_i) \cap proc(t_j) \Longrightarrow eft(t_i, q) < est(t_j, q) \text{ or } eft(t_j, q) < est(t_i, q), \tag{5}$$

that is, the execution of t_i and t_j cannot overlap.

The earliest start time of task t_i on processor p_j is influenced by two parameters, i.e., data ready time (drt) and the earliest idle time block on processor p_j. The data ready time is computed by the following equation:

$$drt(t_i, p_j) = \max_{t_k \in pred(t_i)} \left\{ \max_{p_m \in proc(t_k)} \{eft(t_k, p_m) + cet(e_{k,i}, p_m, p_j)\} \right\}, \tag{6}$$

where $cet(e_{k,i}, p_m, p_j)$ represents the execution time of edge $e_{k,i}$ transferred from processor p_m to p_j.

Condition 3 (Drt Constraint) Let S be a schedule for task graph $G_t = \langle T, E, w, c\rangle$ on a heterogeneous computing system P. For $t_i \in T$ and $p_j \in P$:

$$est(t_i, p_j) = \max\{drt(t_i, p_j), idle(p_j)\}, \tag{7}$$

where $idle(p_j)$ is the earliest idle time block on processor p_j.

Figure 1 gives a DAG example which consists of 10 tasks and 13 edges. Communication costs are marked

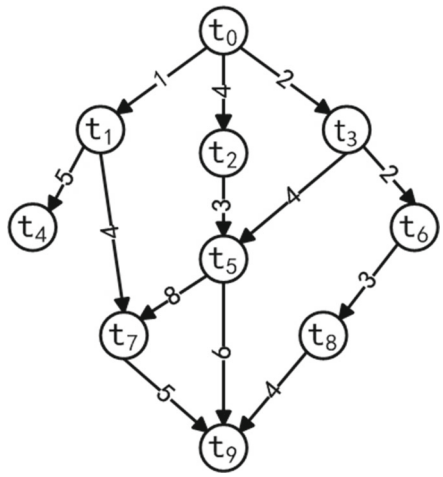


Fig. 1 A DAG example with 10 tasks and 13 edges

in the middle of the edges. Table 2 gives the execution times of the 10 tasks on all the 5 processors.

3.2 Computing System Model

The computing system model used in this paper is heterogeneous. It is modeled as an undirected and partially connected graph $G_p = \langle P, L, F \rangle$, where P denotes the finite set of n heterogeneous processors, L denotes the finite set of undirected communication links, and F denotes the processor frequencies given as a two-dimensional matrix. Let p_j denote the jth processor in set P. And $l_{i,j}$ represents the communication link between processors p_i to p_j. According to the former description, we clearly know that $l_{i,j}$ and $l_{j,i}$ represent the same link. A simple path L between two processors p_s and p_d is defined to be a set of resources in which a resource does not appear more than once. That resource set includes both the source and destination machines as well. This is the definition of the topology which is employed in [8, 26, 38]. Note that this model of the network assumes no Ethernet type of network connections among the processors, which is the only limitation of the model. The speed $s(l_{i,j})$ assigned to a link $l_{i,j}$ represents its communication capacity (the amount of data that can be transmitted on the link in a unit time). We further assume that there is no contention in the communication links. The communication between two tasks executed on the same processor is zero. Figure 2 shows an example of a heterogeneous computing system connected with an arbitrary network.

Traditionally, the failure of a processor and a network is assumed to follow a Poisson distribution with a constant failure rate. According to the existing literature [14, 25, 42], we assume that the faults that occur during the execution of different tasks are independent. $\lambda_{i,j}$ denotes the failure rate of processor p_j when executing task t_i. It is tightly connected to the current processor frequency [38]. Therefore, the failure rate of processor p_j when its frequency is $f_{i,j}$ can be computed by the follow equation [38]:

$$\lambda_{i,j} = \lambda(f_{i,j}) = \lambda_i \cdot g(f_{i,j}) = \lambda_i \cdot 10^{\frac{d(f_{max_i} - f_{i,j})}{f_{max_i} - f_{min_i}}}, \quad (8)$$

where λ_i means the average failure rate of task t_i when the frequency is equal to f_{max_i}. The exponent $d > 0$ is a constant indicating the sensitivity of failure rates to voltage and frequency scaling. f_{min_i} denotes the minimum frequency of task t_i. Similarly, f_{max_i} denotes the maximum frequency of task t_i. In this paper, we assume that f_{max_i} is normalized as 1.0. This failure

Table 2 Execution Time Matrix

Task node	p_0	p_1	p_2	p_3	p_4	Rank(t_i)	Seq
t_0	5	5	4	6	10	41.0889	1
t_1	7	6	9	7	6	24.0000	5
t_2	9	6	4	4	9	33.3111	3
t_3	4	7	9	5	9	34.1556	2
t_4	9	7	8	7	6	7.4000	9
t_5	5	6	6	10	7	25.5778	4
t_6	9	9	7	5	7	22.9111	6
t_7	8	4	8	7	4	15.2222	7
t_8	6	8	4	6	4	14.1778	8
t_9	8	8	6	5	7	6.8000	10

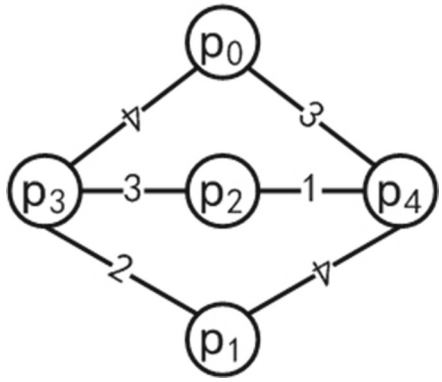


Fig. 2 A heterogeneous computing system with arbitrary networks graph

rate model is consistent with the model presented in [38]. The range of frequency in [38] is from 0.4 to 1.0, which means that the maximum frequency is equal to one. The failure rate of communication link $l_{i,j}$ is denoted as $\lambda(l_{i,j})$, which is not related to frequency.

It should be noted that simulating the failure of a resource by a Poisson process may not always accord with the actual failure dynamics of the resource. However, it has been illustrated by [18] that such an assumption is still useful. 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 rest of the application execution. Note that these assumptions on failures are widely used to deal with the reliability of HC systems [14, 19, 23].

4 The Analysis of Reliability

For a task $t_j \in T$ executed to a processor p_d to run successfully in a heterogeneous computing system, the processor p_d must be operational during the period of task execution and its coming data must be transferred successfully. Assume that task t_i, which is the immediate predecessor of task t_j, is executed to processor p_s. Hence, there exists an operating path from processor p_s to p_d. For current task t_j, its immediate predecessors are more than one, and each of them has some replicas. Hence, searching optimal reliability communication paths for current task is effective for acquiring maximum communication reliability. In this section, we first discuss task reliability and give some useful formulas. Then, we propose an improved algorithm for searching optimal reliability communication paths.

The reliability of a processor p in time interval t is $exp(-\lambda t)$, where λ is the failure rate of processor p. According to this formula, the reliability $r(t_i, p_j)$ of processor p_j during the execution of task t_i, which does not consider precedence constraints, is computed by:

$$r(t_i, p_j) = exp(-\lambda_{i,j} \cdot et(t_i, p_j)). \tag{9}$$

This formula does not consider the precedence between tasks. In other words, we only consider task t_i independently In our reliability model, the failure of the network is assumed to follow a Poisson distribution and each link is associated with two parameters. One is $\lambda(l_{i,j})$ which is the failure rate of link $l_{i,j}$. The other is $s(l_{i,j})$ which is the speed of link $l_{i,j}$.

For the task t_j assigned to p_d and its immediate predecessor task t_i assigned to p_s, let $< l_1, l_2, ..., l_k >$ be the path from p_s to p_d with k links in an HC system. Task t_i may have more than one replica. Therefore, the communication edge $e_{i,j}$ may be transferred more than one time. The earliest start time of link l_u ($u = 1, 2, ..., k$) for edge $e_{i,j}$ from p_s to p_d is denoted as $lest(e_{i,j}, p_s, p_d, l_u)$. Similarly, the earliest finish time of link l_u for edge $e_{i,j}$ from p_s to p_d is denoted as $left(e_{i,j}, p_s, p_d, l_u)$. The execution time of link l_u is denoted as $let(e_{i,j}, p_s, p_d, l_u)$. These three definitions can be described as follows:

$$\begin{cases} lest(e_{i,j}, p_s, p_d, l_u) = eft(t_i, p_s), \\ left(e_{i,j}, p_s, p_d, l_u) = lest(e_{i,j}, p_s, p_d, l_u) + c_{i,j}/s(l_u), \\ let(e_{i,j}, p_s, p_d, l_u) = left(e_{i,j}, p_s, p_d, l_u) - lest(e_{i,j}, p_s, p_d, l_u). \end{cases} \tag{10}$$

Furthermore, we can compute the reliability of communication edge $e_{i,j}$ on link l_u using the following equation:

$$r(e_{i,j}, p_s, p_d, l_u) = exp(-\lambda(l_u) \cdot let(e_{i,j}, p_s, p_d, l_u)). \tag{11}$$

Since the failure of network links are statistically independent. From the above equation, the reliability communication edge $e_{i,j}$ can be computed by:

$$r(e_{i,j}, p_s, p_d) = \prod_{u=1}^{k} r(e_{i,j}, p_s, p_d, l_u). \tag{12}$$

Communication contention is not considered in this paper. So the execution time of communication edge $e_{i,j}$ is computed as follows:

$$cet(e_{i,j}, p_s, p_d) = \sum_{u=1}^{k} let(e_{i,j}, p_s, p_d, l_u). \quad (13)$$

It is easy to obtain the earliest start and finish times of $e_{i,j}$ according to the above argument:

$$\begin{cases} cest(e_{i,j}, p_s, p_d) = lest(e_{i,j}, p_s, p_d, l_1), \\ ceft(e_{i,j}, p_s, p_d) = left(e_{i,j}, p_s, p_d, l_k). \end{cases} \quad (14)$$

According to the analysis given above, we can define the present reliability of task t when it is executed to processor p.

Definition 1 Let $pr(t_j, p_z)$ denote the task present reliability. In other words, it is the probability that no failure occurs on processor p_z from the beginning of the schedule to the end of task t_j's execution on p_z. It is influenced by the present reliability of its immediate predecessor tasks and the communication reliability. This value can be acquired by:

$$pr(t_k, p_j) = \prod_{t_i \in pred(t_k)} \left(1 - \prod_{p_n \in proc(t_i)} (1 - pr(t_i, p_n) \cdot r(e_{i,j}, p_n, p_j))\right) \cdot r(t_k, p_j). \quad (15)$$

There are differences between $pr(t_j, p_z)$ and $r(t_j, p_z)$. The time span of $pr(t_j, p_z)$ ranges from the beginning of a schedule to the end of the execution of task t_j. Hence, it is crucial to consider the reliability generated from the predecessors of t_j. However, the time span of $r(t_k, p_j)$ ranges from the beginning of the execution of task t_j on processor p_z to the end of that. As a result, it only considers task t_j without considering its predecessors.

In order to maximize task present reliability, we propose an improved algorithm called OPMR(t_k, p_j, E, P, pr) which enhances the communication reliability for task t_k. OPMR selects the maximum reliability paths from processor p_j to all the processors on which the immediate predecessors of task t_k are executed. The pseudo code of OPMR is shown in Algorithm 1. Firstly, a vector $r[z]$ which stores the maximum communication reliability value from processor p_j to all the other processors is settled and initialized (lines 1–5). Secondly, we use a structure to save all the optimal reliable paths from a processor to another (line 6). For any path, the processors are saved orderly. Thirdly, we repeatedly select the processor p_u in set $P - S$ which has the maximum communication reliability value until the set $P - S$ is empty. Then, we recompute the communication reliability value of all links transferred to p_j through p_u. If the new value is larger than the former value, then we replace it (lines 7–21). Lastly, the present reliability of task t_k on processor p_j can be computed (lines 22–23).

The network failure rates are given in Table 3, which is based on the topology in Fig. 2. Assuming that task t_0 is scheduled on processor p_0 and task t_2 is scheduled on processor p_1. According to Fig. 2, there are three available communication path between processor p_0 and p_1. The optimal reliable communication path found by the OPMR algorithm is $l_{0,3}$ and $l_{1,3}$.

5 The Proposed Algorithm

This section proposes a new algorithm named RMSR (Replication-based scheduling for Maximizing System Reliability) in HC systems. This algorithm consists of two phases. One is the listing phase, which is similar to the HEFT algorithm [31]. The other is the task replication and assignment phase, which maximizes the system reliability. RMSR is an improved algorithm for the existing reliability-aware algorithms. We will describe it in two steps in the following subsections. The pseudo code of RMSR is shown in Algorithm 2.

5.1 Task Prioritizing Phase

In our RMSR algorithm, tasks are ranked by their priorities which are based on bottom-level. It is calculated recursively by

$$rank(t_i) = \overline{et(t_i)} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank(t_j)), \quad (16)$$

Algorithm 1 OPMR(t_k, p_j, E, P, pr)

Input: Task t_k when it is executed on processor p_j, the set of communication edges E, the set of processors P, the present reliability value pr of all t_k's predecessors.

Output: The present reliability of task t_k when it is executed on p_j.

1. **for** each task $t_i \in pred(t_k)$ **do**
2. Put all the processors in set $proc(t_i)$ into set T, and put p_j into set S;
3. Set a vector $r[z]$ which represents the max communication reliability value form p_z to p_j, $p_z \in P$;
4. Initialize all processors in set P;
5. Compute vector $r[z]$ by

$$r[z] = \begin{cases} r(e_{i,j}, p_z, p_j, l_1) \text{ using Eq. (11)} & \text{if } p_j \text{ connects } p_z; \\ 1 & \text{if } p_j = p_z; \\ 0 & \text{otherwise;} \end{cases}$$

6. Set a two-dimensional matrix structure path[j][z] to save optimal reliable path from p_z to p_j;
7. **while** $T \neq \emptyset$ **do**
8. Select a processor p_u which has the max $r[z]$ in set $P - S$ and put it into set S;
9. **for all** processor p_z in set $P - S$ **do**
10. Compute the reliability rl of link transferred from p_z to p_j through p_u using Eq. (12);
11. **if** $rl > r[z]$ **do**
12. $r[z]$=rl;
13. Put processor p_u in path[j][z];
14. **end if**
15. **end for**
16. **if** $p_u \in T$ **then**
17. Remove p_u from T;
18. **end if**
19. **end while**
20. $r(e_{i,j}, p_z, p_j) = r[z]$;
21. **end for**
22. Compute $pr(t_k, p_j)$ according to Eq. (15);
23. **return** $pr(t_k, p_j)$.

Algorithm 2 The RMSR Algorithm

Input: A DAG graph $G_t = \langle T, E, w, c \rangle$, task reliability threshold γ, a processor topology graph $G_p = \langle P, L, F \rangle$.

Output: A task schedule S of G_t on G_p.

1. **for all** task t_i in G_t **do**
2. **for all** processor p_j in G_p **do**
3. Calculate $et(t_i, p_j)$ using Eq. (1);
4. Compute $r(t_i, p_j)$ using Eq. (9);
5. **end for**
6. **end for**
7. Compute the rank for each task using Eq. (16);
8. Sort all the task in non-increasing order according to rank, then put them in queue NT in order;
9. **while** $NT \neq \emptyset$ **do**
10. Select the first task t_k in NT;
11. **for all** processor p_j **do**
12. Compute $pr(t_k, p_j)$ using Algorithm 1;
13. **end for**
14. Sort all the processor p_j in decreasing order according to $pr(t_k, p_j)$, then put them in queue NP_k in order;
15. Select the first processor p_n on NP_k., and assume its pr value is pr_{max};
16. **if** $pr_{max} > \gamma$ **then**
17. $proc(t_k) = p_n$;
18. **else**
19. Compute the number $\beta_{num}(t_k)$ of replicas that task t_k needs and determine the set $proc(t_k)$ of processors on which the replicas execute using Algorithm 3;
20. **end if**
21. **for all** processor $p_j \in proc(t_k)$ **do**
22. Calculate $est(t_k, p_j)$ using Eq. (7);
23. Schedule t_k on p_j;
24. **end for**
25. **end while**
26. **return** S.

where $\overline{et(t_i)}$ is the average computation time of task t_i and $\overline{c_{i,j}}$ is the average communication cost of edge $e_{i,j}$ over all paths between any two processors. The rank is computed recursively by traversing the task upward, starting from the exit task. For each exit task t_{exit}, the rank value is computed by:

$$rank(t_{exit}) = \overline{et(t_{exit})}. \quad (17)$$

In Algorithm 2, this phase is shown in lines 1–8. Firstly, two parameters $et(t_i, p_j)$ and $r(t_i, p_j)$ are computed in lines 1–6. Secondly, we compute the rank for each task recursively (line 7). Lastly, all the tasks are sorted in non-increasingorder according to their ranks, and consequently a task schedule list which satisfies inter-task precedence is generated (line 8). For

Table 3 Network Failure Rate

Link	$l_{0,3}$	$l_{0,4}$	$l_{1,3}$	$l_{1,4}$	$l_{2,3}$	$l_{2,4}$
Failure rate	0.0010	0.0020	0.0004	0.0005	0.0012	0.0015

example, considering the application DAG in Fig. 1 and the heterogeneous computing system in Fig. 2, the rank values which are computed by (16) and the task sequence are listed in Table 2.

5.2 Task Replication and Assignment Phase

In this phase, the replication number of task t_i is determined and tasks are allocated to the suitable processors. A few terms related to reliability are defined for reader's convenience. The definitions are given below.

Definition 2 The task reliability threshold is denotes as γ. It is the lower bound of task reliability.

Definition 3 The task final reliability $fr(t_i)$ is the reliability of task t_i after replication. After tasks are allocated to some suitable processors, it can be computed by the following equation:

$$fr(t_i) = 1 - \prod_{p_j \in proc(t_i)} (1 - pr(t_i, p_j)). \tag{18}$$

Definition 4 The system reliability R is the product of the final reliability of all the sink tasks, which can be calculated by using the following equation:

$$R = \prod_{succ(t_i)=\emptyset} fr(t_i). \tag{19}$$

The main process of this phase is shown in lines 9–26 of Algorithm 2. According to the task sequence generated in Section 5.1, for the current task t_k, we compute $pr(t_k, p_j)$ for each processor p_j using Algorithm 1 (lines 10–13). Then we sort all the processors in non-increasing order according to the pr values and put this order in the queue NP_k. It is obvious to know that the first processor in NP_k has the largest pr value. We denote that processor as p_n and the corresponding pr value as pr_{max} (lines 14–15). To guarantee that the final reliability of task t_k is greater than the task reliability threshold γ, we compare pr_{max} to γ. If pr_{max} is higher than γ, the replication of task t_k is not needed. As a result, $\beta_{num}(t_k)$ is equal to one and consequently we choose processor p_n for task t_k to be allocated. But if pr_{max} is lower than γ, the replication of task t_i is needed. We use Algorithm 3 to compute $\beta_{num}(t_k)$ and generate $proc(t_k)$ at the same time (lines 16–20). As a result, the replication number of each task is dynamic. After the replication process is completed, we assign task t_k to all the processors in $proc(t_k)$ (lines 21–24).

In Algorithm 3, we dynamically compute the replication number of task t_k according to its present reliability (pr) and task reliability threshold γ. Firstly, the first two processors in NP_k are selected and the computation of $fr(t_k)$ is executing as an initial value (lines 1–5). Secondly, we repeatedly select the next processor in NP_k, recompute $fr(t_k)$ with (18) until $fr(t_k)$ is larger than γ (lines 6–11). Lastly, the replication number of task t_k is saved in $\beta_{num}(t_k)$ (lines 12–13).

Algorithm 3 Repnum(t_k, pr, NP_k, γ, n)

Input: A given task reliability threshold γ, the task t_i which needs to compute its replicas, all the reliability values of t_i on P, a sorted processor set NP_k according to pr, the processor number n.

Output: The number of replicas for tasks t_k – $\beta_{num}(t_k)$.

1. num=1;
2. Select the first two processors in NP_k – NP_k^0 and NP_k^1;
3. $r = 1-(1-pr(t_k, NP_k^0)) \times (1-pr(t_k, NP_k^1))$;
4. $p = 1 - r$;
5. $proc(t_k) = NP_k^0$;
6. **while** $1 - p < \gamma$ and $num < n$ **do**
7. Put NP_k^{num} into $proc(t_k)$;
8. num++;
9. $r = pr(t_k, NP_k^{num})$;
10. $p = p \times (1 - r)$;
11. **end while**
12. $\beta_{num}(t_k) = num$;
13. **return** $\beta_{num}(t_k)$ and $proc(t_k)$.

5.3 Time Complexity of RMSR

The time complexity of the RMSR algorithm is expressed in terms of the number of nodes $|T|$, the number of processors $|P|$, and the maximum degree of each task d_{in}^{max}. The time complexity of RMSR is analyzed in two steps. Firstly, we analyze the complexity of two subalgorithms, i.e., the OPMR and Repnum algorithms. Then, we describe the process of analyzing the complexity of RMSR according to the values acquired from the first step.

In the OPMR algorithm (Algorithm 1), all predecessors of each task are considered, resulting in time $O(d_{in}^{max})$. For each predecessor of any task, the following process is repeated. Initializing all the processor and searching optimal reliable paths can be done in time $O(|P|)$ and $O(|P|^2)$. As a result, the overall complexity of OPMR is $O(d_{in}^{max}|P|^2)$. In the Repnum algorithm (Algorithm 3), the replication number of task t_k is computed and denoted as $\beta_{num}(t_k)$, which is lower than $|P|$. The overall complexity of Repnum is $O(\beta_{num}(t_k))$.

In each round of the RMSR algorithm (Algorithm 2), calculating execution time and reliability of each task on all processors can be done in time $O(|T||P|)$. Computing the rank value and sorting the tasks can be done in time $O(|T|\log_2|T|)$. Task replication and assignment phase can be done in time $O(|T||P|^3 d_{in}^{max})$. Hence, the complexity of RMSR is $O(|T|\log_2|T| + |T||P|^3 d_{in}^{max})$.

6 Experimental Results and Analysis

In this section, we compare the performance of the proposed RMSR algorithm with that of the existing scheduling algorithms in heterogeneous computing system: the RASD [26] and the HEFT [31] algorithms. For RMSR, we list two cases when the task reliability thresholds γ is equal to 0.995 and 0.99, respectively. For convenience of description, we mark task reliability threshold value behind RMSR in the bracket. To make effective comparison, we consider two sets of graphs as the workload for testing the algorithms, i.e., randomly generated application graphs and graphs that represent some of the real-world numerical problems. The two real-world parallel applications used for our experiments are the Gaussian elimination algorithm [6, 16, 31, 35] and the fast Fourier transformation algorithm [5, 31]. System reliability is our comparison metric to test the performance of these algorithms.

To test the performance of these algorithms, we have built an extensive simulation environment of HDC systems with 32 processors whose computation capacities vary from Pentium II to Pentium IV. In order to make the three algorithms comparable, we modified the computing system model of HEFT properly according to our models presented in Section 3.

6.1 Randomly Generated Applications

Generation of Random Application Graphs In our study, we first considered randomly generated application graphs. A random graph generator was implemented to generate weighted application DAGs with various characteristics that depend on several input parameters [8, 16, 26, 31]. It requires the following input parameters to build weighted DAGs.

- Number of DAG nodes m.
- Parallelism parameter α. We assume that the height of a DAG h is randomly generated from a uniform distribution with a mean value equal to $\frac{\sqrt{m}}{\alpha}$. (The height is equal to the smallest integral value not less than the real value generated randomly). The width for each level is randomly selected form a uniform distribution with a mean equal to $\alpha\sqrt{m}$.
- Out degree of a node, *out_degree*.
- Communication to computation time ratio CCR. It is the ratio of the average communication cost to the average computation cost. A DAG can be considered as a computation-insensitive application if its CCR value is very low.
- The average computation cost of each task $\overline{w_i}$. It is selected randomly from a uniform distribution with range $[0, 2\overline{w_{DAG}}]$, where $\overline{w_{DAG}}$ is the average computation cost of the given graph. The value of $\overline{w_{DAG}}$ does not affect the performance results of the scheduling algorithms.
- The frequency f ranges from 0.4 to 1.

In our simulation experiments, DAGs are generated based on the combination of parameters introduced above. The number of tasks in a DAG ranges from 40 to 200. To determine the size of a generated DAG, the height of this DAG is computed by parallelism parameter α (0.5, 1.0, 2.0) firstly, then the width of each level

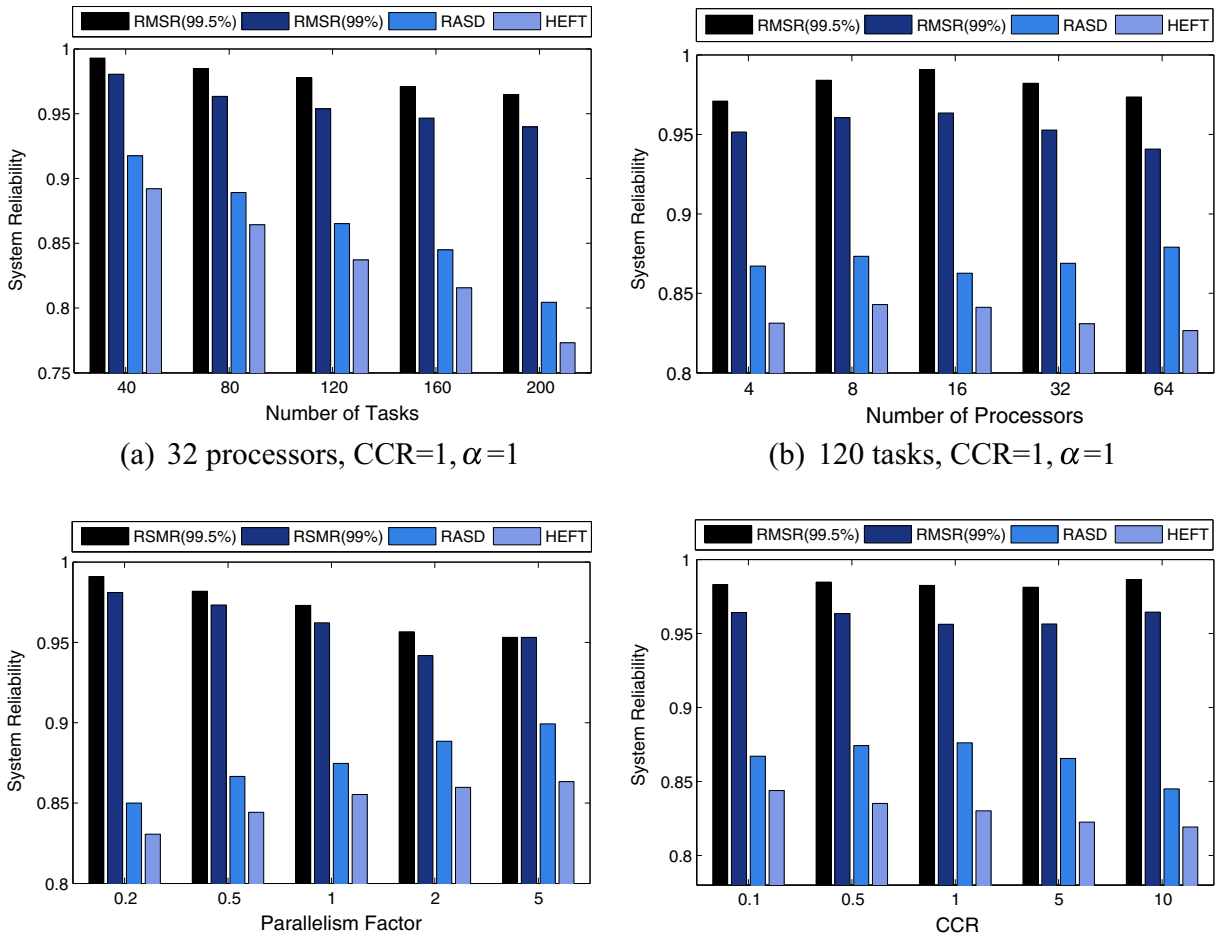
is determined. To obtain the desired CCR for a DAG, the average computation cost $\overline{w_{DAG}}$ is randomly from a uniform distribution. The communication costs are also taken from a uniform distribution with a mean which equals to the product of $\overline{w_{DAG}}$ and CCR (0.1, 0.5, 1.0, 5.0, 10.0). 900 random graphs are generated for each set of above parameters in order to avoid scattering effects. The experimental results are the average of the data obtained for these DAGs.

Random Applications Performance Analysis The system reliability of three algorithms is compared with respect to graph characteristics. The overall experimental results are presented in Fig. 3, which gives an intuitive presentation in two aspects. Firstly, compared to RASD and HEFT, our RMSR algorithm achieves noticeable improvement on system reliability. Secondly, with the increasing of task reliability threshold γ, the system reliability in our algorithm increases correspondingly.

The first set of experiments compare system reliability with respect to the number of tasks (see Fig. 3a). The performance of our RMSR algorithm outperforms both RASD and HEFT algorithms, especially when the number of tasks is large. According to Fig. 3a, the system reliability of RMSR(99.5 %) is greater than RASD and HEFT by (7.53 %, 10.08 %), (9.54 %, 12.04 %), (11.24 %, 14.05 %), (12.59 %, 15.53 %), and (16.03 %, 19.15 %), for the number of tasks of 40, 80, 120, 160, and 200, respectively. It shows that the system reliability decreases with the increasing number of tasks. The reason is described as follows. When the task number increases, the number of sink tasks is increasing correspondingly. According to the calculation formula for system reliability in (19), the more sink tasks are, the lower system reliability can be acquired. Our algorithm uses replication to guarantee that the final reliability for each task is higher than γ. As a result, this method can prevent the decreasing of system reliability as much as possible. There is another phenomenon presented in Fig. 3a. For the same set of DAGs, with the increasing of task reliability threshold γ, the system reliability increases for the two cases of RMSR. This is because the higher the task reliability threshold is, the higher task final reliability can get. As a result, the system reliability is also higher correspondingly.

The second set of experiments compare system reliability of the four cases with respect to different numbers of processors. As shown in Fig. 3b, RMSR(99.5 %) significantly outperforms RASD and HEFT by (10.38 %, 13.97 %), (11.08 %, 14.11 %), (12.78 %, 14.93 %), (11.32 %, 15.13 %), (9.45 %, 14.69 %), for number of processors of 4, 8, 16, 32, 64, respectively. When the number of processors is lower than 16, the system reliability of the two cases of RMSR slightly increases. However, when the number of processors is higher than 16, the system reliability slightly decreases. Therefore, all the two cases of RMSR choose 16 as the most suitable processor number. The possible reasons resulting in this phenomenon are explained as follows. On the one hand, with the increasing number of processors, the average number of links from one processor to another is relatively increased. Hence, link reliability decreases and influences the system reliability indirectly. On the other hand, when the number of processors is small, if the failure rate of some processors is large, this will influence all the tasks allocated to them. The system reliability is also influenced indirectly. According to the analysis given above, we assume that when task number is fixed, there exists a suitable processor number. And for different task numbers, different suitable processor numbers can be found.

In the third set of experiments, the results are depicted in Fig. 3c for all tested parallelism factor values. With the increasing parallelism factor value, the results show three different trends for all the four cases. The explanation is described as follows. For RASD and HEFT, when the number of tasks is fixed, the height of a DAG varies inversely with the parallelism factor value. In other words, the smaller the parallelism factor value is, the greater the number of predecessors a task has, hence leading to lower system reliability. Because of the replication method we use, the phenomenon RASD shows can not affect our algorithm. For RMSR, the system reliability is influenced by two parameters. One is the number of sink tasks. The other is the final reliability of sink tasks. When the parallelism factor value ranges from 0.2 to 5, the number sink nodes becomes larger for all the two cases in RMSR. However, with the increasing task reliability threshold, the range of final reliability of sink tasks is less. Therefore, the influence of final reliability of sink tasks is little. We believe that the results of RMSR(99.5 %) are only influenced by the first parameter, while the results of RMSR(99.0 %)



(a) 32 processors, CCR=1, α=1

(b) 120 tasks, CCR=1, α=1

(c) 120 tasks, 32 processors, CCR=1

(d) 120 tasks, 32 processors, α=1

Fig. 3 Average system reliability of random DAGs

are influenced by all the two parameters. In short, our algorithm outperforms RASD and HEFT significantly when the parallelism factor is small.

The last set of experiments are with respect to the CCR. As shown in Fig. 3d, the CCR value does not significantly affect the relative performance of scheduling algorithms. In other words, the performance comparison shows similar patterns regardless of CCR. The pattern shows that for the same set of DAGs, the two cases of our algorithm provide higher system reliability than RASD and HEFT by the average value 11.05 % and 15.32 %.

6.2 Application Graphs of Real-World Problems

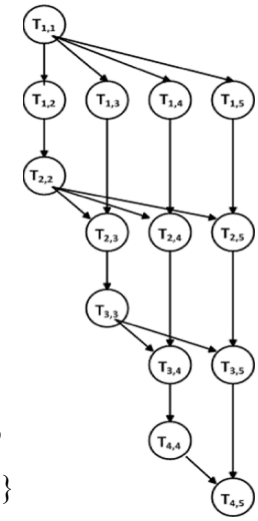
In addition to randomly generated task graphs, we also considered application graphs of real-world problems, i.e., the Gaussian elimination [6, 16, 31, 35] and the fast Fourier transformation (FFT) [31, 35].

Gaussian elimination Gaussian elimination is used to determine the solution of a linear system of equations. Figure 4a gives the sequential program for Gaussian elimination algorithm. The DAG of the algorithm solving a 5×5 matrix is shown in Fig. 4b. For the experiments of Gaussian elimination application, the same CCR and number of processors were used. Since the structure of the application graph is known, we do not need other parameters, such as task number, *out_degree*, and parallelism factor α. A new parameter, matrix size z, is used in place of DAG size m (the number of tasks in the DAG graph). The total number of tasks in a Gaussian elimination graph is equal to $\frac{1}{2}(z^2 + z - 2)$ [35].

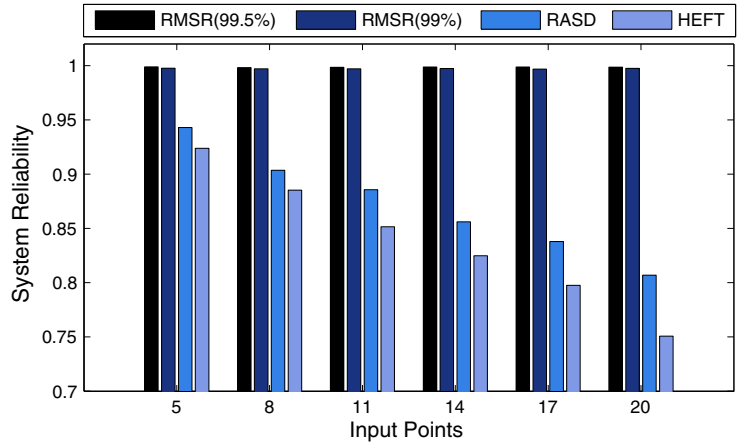
Fig. 4 **a** Gaussian elimination algorithm; **b** Task graph for matrix of size 5; **c** System reliability comparison for Gaussian elimination graph

$$
\begin{aligned}
&for \quad k=1 \quad to \quad m-1 \quad do \\
&\quad T_{k,k}:\{\ for \quad i=k+1 \quad to \quad m \quad do \\
&\qquad\qquad a_{ik}=a_{ik}/a_{kk} \qquad \} \\
&\quad for \quad j=k+1 \quad to \quad m \quad do \\
&\qquad T_{k,j}:\{\ for \quad i=k+1 \quad to \quad m \quad do \\
&\qquad\qquad a_{ij}=a_{ij}\cdot a_{ik}\cdot a_{kj} \qquad \}
\end{aligned}
$$

(a)



(b)



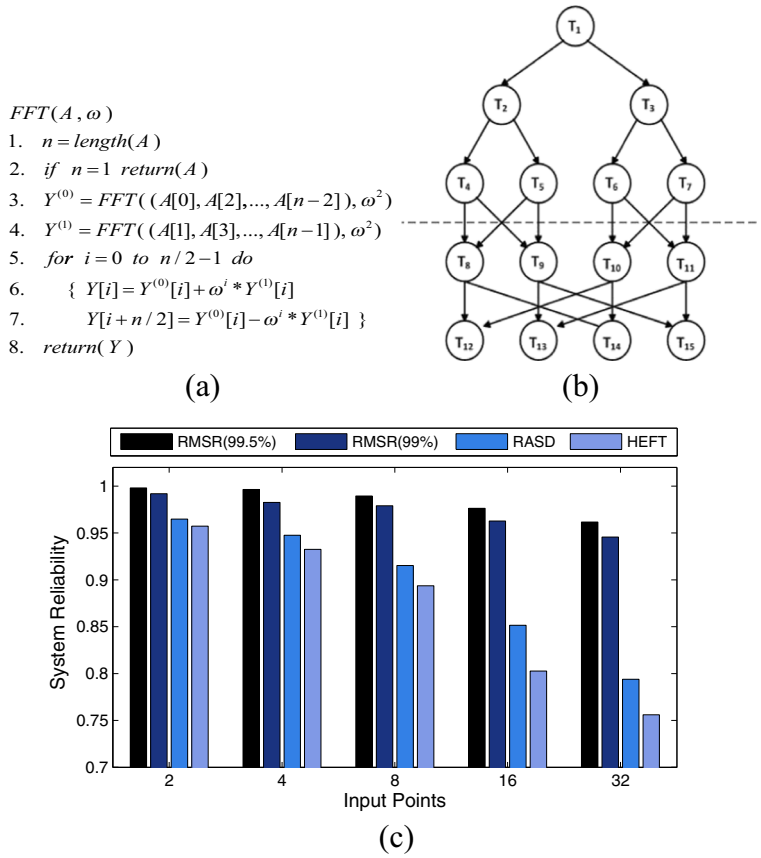
(c)

Figure 4c shows the system reliability of the four cases at various matrix sizes from 5 to 20, with an increment of three, when the number of processors is equal to eight. The smallest size graph in this experiment has 14 tasks and the largest one has 209 tasks. With the increasing of matrix size, the results of all the two cases of RMSR change slightly. The reason is that the DAG of Gaussian elimination application has only one sink task. According to our description before, system reliability is equal to the final reliability of this sink task in this situation. Therefore, the system reliability is larger than γ. Overall, our algorithm RMSR(99.5 %) outperforms RASD and HEFT by (5.60 %, 7.50 %), (9.45 %, 11.28 %), (11.27 %, 14.70 %), (14.3 %, 17.39 %), (16.07 %, 20.11 %), (19.15 %, 24.78 %), for matrix sizes 5, 8, 11, 14, 17, 20, respectively.

Generally speaking, when the number of sink tasks of a DAG is only one, we can guarantee that the lower bound of system reliability is larger than task reliability threshold γ.

Fast Fourier transformation (FFT) The recursive, one-dimensional FFT algorithm [31, 35] and its task graph (when there are four data points) are given in Fig. 5. In this figure, A is an array of size z which holds the coefficients of the polynomial, and array Y is the output of the algorithm. The behavior of FFT with input vector size 4 is shown in Fig. 5a. The algorithm consists of two parts: recursive calls

Fig. 5 **a** FFT algorithm; **b** The generated DAG of FFT with 4 points; **c** System reliability comparison for FFT graph



(lines 3–4) and the butterfly operation (lines 6–7). The task graph in Fig. 5b can be divided into two parts, the tasks above the dashed line are the recursive call tasks and the ones below the line are butterfly operation tasks. For an input vector size of z, there are $2z - 1$ recursive call tasks and $z \log_2 z$ butterfly operation tasks. (We assume that $z = 2^k$ for some integer k.)

When the input vector size is known, the number of sink nodes of a DAG is fixed correspondingly. Therefore, the system reliability is affected by the final reliability of each sink task. As a result, the experiment results of RMSR are only influenced by the input vector size regardless of other parameters through replication. The comparison of system reliability of the four cases is shown in Fig. 5c. It is obvious to see that our algorithm significantly outperforms both RASD and HEFT algorithms. For example, the system reliability of RMSR(99.5 %) is greater than RASD and HEFT by (3.32 %, 4.07 %), (4.88 %, 6.37 %), (7.42 %, 9.58 %), (12.47 %, 17.35 %), (16.78 %, 20.55 %), for the input vector size of 2, 4, 8, 16, 32, respectively. This provides a clear indication that there is a trend of improved performance with increasing input vector size.

7 Conclusion

In this paper, we have proposed a new reliability-aware task scheduling algorithm called RMSR in heterogeneous computing systems. The aim of this algorithm is to maximize the system reliability based on replication. A task reliability threshold is used when the replication executes. The performance of the RMSR algorithm is compared to two of the best existing scheduling algorithms for HC systems, i.e., the HEFT and RASD algorithms. Because the system reliability achieved by our algorithm is tightly related to the task reliability threshold, we set two different task reliability threshold, 99.5 % and 99.0 %, to show the different performance of RMSR.

The performance comparison is based on both randomly generated application DAGs and two real-world problems, namely, Gaussian elimination and fast Fourier transformation (FFT). The experimental results show that the two cases of our algorithm outperforms both HEFT and RASD in terms of system reliability. We can also find that for some particular shapes of DAG whose number of sink tasks is one, the system reliability of our algorithm can be guaranteed by the task reliability threshold. Furthermore, for a DAG whose number of sink tasks is more than one, the system reliability of our algorithm is only influenced by task numbers. Overall, RMSR makes a huge improvement of system reliability.

This work represents our first and preliminary attempt to study a very complicated problem. Further study in this area may take the energy saving issue into account.

Acknowledgments A preliminary version of the paper will be presented in the *Fourth Workshop on Parallel Computing and Optimization* in conjunction with IPDPS 2014, Phoenix, Arizona, May 23, 2014. The authors would like to thank the three anonymous reviewers for their valuable and helpful comments on improving the manuscript.

References

1. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2014)
2. Benoit, A., Hakem, M., Robert, Y.: Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In: IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, pp. 1–8 (2008)
3. Bittencourt, L.F., Sakellariou, R., Madeira, E.R.: Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In: 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 27–34. IEEE (2010)
4. Chiu, C.C., Yeh, Y.S., Chou, J.S.: A fast algorithm for reliability-oriented task assignment in a distributed system. Comput. Commun. **25**(17), 1622–1630 (2002)
5. Chung, Y.C., Ranka, S.: Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In: Supercomputing'92., Proceedings, pp. 512–521. IEEE (1992)
6. Daoud, M.I., Kharma, N.: A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **68**(4), 399–409 (2008)
7. Dogan, A., Ozguner, F.: Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing. In: Proceedings of the 2000 International Workshops on Parallel Processing, 2000, pp. 429–436. IEEE (2000)
8. Dogan, A., Ozguner, F.: Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 308–323 (2002)
9. Dongarra, J.J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures, pp. 280–288. ACM (2007)
10. El-Rewini, H., Lewis, T.G.: Scheduling parallel program tasks onto arbitrary target machines. J. Parallel Distrib. Comput. **9**(2), 138–153 (1990)
11. Elegbede, A.C., Chu, C., Adjallah, K.H., Yalaoui, F.: Reliability allocation through cost minimization. IEEE Trans. Reliab. **52**(1), 106–111 (2003)
12. Hsieh, C.C.: Optimal task allocation and hardware redundancy policies in distributed computing systems. Eur. J. Oper. Res. **147**(2), 430–447 (2003)
13. Iverson, M.A., Özgüner, F., Follen, G.J.: Parallelizing existing applications in a distributed heterogeneous environment. In: 4TH Heterogeneous Computing Workshop (HCW'95. Citeseer (1995)
14. Kartik, S., Murthy, C.S.R.: Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans. Comput. **46**(6), 719–724 (1997)
15. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. J. Parallel Distrib. Comput. **59**(3), 381–422 (1999)
16. Liu, G., Poh, K.L., Xie, M.: Iterative list scheduling for heterogeneous computing. J. Parallel Distrib. Comput. **65**(5), 654–665 (2005)
17. Michael, R.G., Johnson, D.S.: Computers and intractability: A guide to the theory of np-completeness. WH Freeman &Co., San Francisco (1979)
18. Plank, J.S., Elwasif, W.R.: Experimental assessment of workstation failures and their impact on checkpointing systems. In: Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers. pp. 48–57. IEEE (1998)
19. Qin, X., Jiang, H.: A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. J. Parallel Distrib. Comput. **65**(8), 885–900 (2005)
20. Raj, J.S., Rachel, I.S.: A survey on reliability scheduling on grid computing. In: 2013 7th International Conference on Intelligent Systems and Control (ISCO), pp. 331–334. IEEE (2013)
21. Ramírez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. Journal of Grid Computing **9**(1), 95–116 (2011)
22. Sakellariou, R., Zhao, H.: A hybrid heuristic for dag scheduling on heterogeneous systems. In: Proceedings. 18th International Parallel and Distributed Processing Symposium, 2004. p. 111. IEEE (2004)
23. Shatz, S.M., Wang, J.P., Goto, M.: Task allocation for maximizing reliability of distributed computer systems. IEEE Trans. Comput. **41**(9), 1156–1168 (1992)

24. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Trans. Parallel Distrib. Syst. **4**(2), 175–187 (1993)
25. Srinivasan, S., Jha, N.K.: Safety and reliability driven task allocation in distributed systems. IEEE Trans. Parallel Distrib. Syst. **10**(3), 238–251 (1999)
26. Tang, X., Li, K., Li, R., Veeravalli, B.: Reliability-aware scheduling strategy for heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **70**(9), 941–952 (2010)
27. Tang, X., Li, K., Liao, G.: An effective reliability-driven technique of allocating tasks on heterogeneous cluster systems. Clust. Comput., 1–13 (2014)
28. Tang, X., Li, K., Qiu, M., Sha, E.H.M.: A hierarchical reliability-driven scheduling algorithm in grid systems. J. Parallel Distrib. Comput. **72**(4), 525–535 (2012)
29. Tao, Y., Jin, H., Shi, X.: Grid workflow scheduling based on reliability cost. In: Proceedings of the 2nd international conference on Scalable information systems, p. 12. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2007)
30. Tom, P.A., Murthy, C.: Algorithms for reliability-oriented module allocation in distributed computing systems. J. Syst. Softw. **40**(2), 125–138 (1998)
31. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
32. Tosun, S.: Energy-and reliability-aware task scheduling onto heterogeneous mpsoc architectures. J. Supercomput. **62**(1), 265–289 (2012)
33. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
34. Wang, X., Yeo, C.S., Buyya, R., Su, J.: Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm. Futur. Gener. Comput. Syst. **27**(8), 1124–1134 (2011)
35. Wu, M.Y., Gajski, D.D.: Hypertool: A programming aid for message-passing systems. IEEE Trans. Parallel Distrib. Syst. **1**(3), 330–343 (1990)
36. Zhang, J., Gu, J.: An approach to analyze grid service reliability subject to failures. In: 4th International Conference on Computer Science & Education, 2009. ICCSE'09. pp. 343–347. IEEE (2009)
37. Zhang, X., Zagorodnov, D., Hiltunen, M., Marzullo, K., Schlichting, R.D.: Fault-tolerant grid services using primary-backup: feasibility and performance. In: 2004 IEEE International Conference on Cluster Computing, pp. 105–114. IEEE (2004)
38. Zhao, B., Aydin, H., Zhu, D.: On maximizing reliability of real-time embedded applications under hard energy constraint. IEEE Trans. Ind. Inf. **6**(3), 316–328 (2010)
39. Zhao, L., Ren, Y., Sakurai, K.: A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems. In: 2011 IEEE International Conference on Advanced Information Networking and Applications (AINA), pp. 275–282. IEEE (2011)
40. Zhao, L., Ren, Y., Xiang, Y., Sakurai, K.: Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems. In: 2010 12th IEEE International Conference on High Performance Computing and Communications (HPCC), pp. 434–441. IEEE (2010)
41. Zheng, Q., Veeravalli, B., Tham, C.K.: On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs. IEEE Trans. Comput. **58**(3), 380–393 (2009)
42. Zhu, D., Melhem, R., Mossé, D.: The effects of energy management on reliability in real-time embedded systems. In: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004. pp. 35–40. IEEE (2004)