Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems

Kenli Li, Xiaoyong Tang, Bharadwaj Veeravalli, *Senior Member*, *IEEE*, and Keqin Li, *Senior Member*, *IEEE*

Abstract—Generally, a parallel application consists of precedence constrained stochastic tasks, where task processing times and intertask communication times are random variables following certain probability distributions. Scheduling such precedence constrained stochastic tasks with communication times on a heterogeneous cluster system with processors of different computing capabilities to minimize a parallel application's expected completion time is an important but very difficult problem in parallel and distributed computing. In this paper, we present a model of scheduling stochastic parallel applications on heterogeneous cluster systems. We discuss stochastic scheduling attributes and methods to deal with various random variables in scheduling stochastic tasks. We prove that the expected makespan of scheduling stochastic tasks is greater than or equal to the makespan of scheduling deterministic tasks, where all processing times and communication times are replaced by their expected values. To solve the problem of scheduling precedence constrained stochastic tasks efficiently and effectively, we propose a stochastic dynamic level scheduling (SDLS) algorithm, which is based on stochastic bottom levels and stochastic dynamic levels. Our rigorous performance evaluation results clearly demonstrate that the proposed stochastic task scheduling algorithm significantly outperforms existing algorithms in terms of makespan, speedup, and makespan standard deviation.

Index Terms—Directed acyclic graph, heterogeneous cluster system, parallel processing, stochastic task scheduling

1 Introduction

1.1 Motivation

In the past few years, cluster systems have become primary and cost-effective infrastructures for parallel processing. Parallel processing is a promising approach to meet the computational requirements of a large number of current and emerging applications, including information processing, fluid flow, weather modeling, and image processing. The data and computations of such an application can be distributed on the processors of a cluster system, and thus maximum benefits from these systems can be obtained by employing efficient task partitioning and scheduling strategies. More and more evidence show that the scheduling of parallel applications is highly critical to the performance of cluster systems. The common objective of scheduling is to map tasks of a parallel application onto processors of a cluster system and order their executions, so that task precedence constraints are satisfied and the minimum makespan is achieved [1]-[4].

Most classical and existing scheduling researches focus on the deterministic version of the scheduling problem with deterministic computation and communication times [1]-[9]. That is to say, most of the known algorithms assume that parameters such as task processing times and intertask communication times are fixed and deterministic, which are known in advance. However, in many real-world problems, tasks may not have fixed execution times. Such tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs [10]-[15], [34], such as the smoothed particle hydrodynamics computation in interfacial flows numerical simulation [33]. Furthermore, communication times among tasks can also fluctuate according to network traffic.

Although many deterministic scheduling techniques can thoroughly check for the best assignment of tasks to processors, existing methods are not able to deal with randomness and uncertainty effectively. A natural step to tackle this problem is to consider stochastic task scheduling, that is, to interpret processing times and communication times as random variables and to measure the performance of an algorithm by its expected objective value. In this paper, we let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of n precedence constrained tasks that need to be scheduled on a cluster system with m heterogeneous processors, so as to minimize the makespan. A processor can process at most one task at a time, and every task has to be processed on one of the m processors. In contrast to deterministic tasks, an important assumption for stochastic tasks is that the task processing time $w(v_i)$ of v_i is not known in advance. Instead, one assumes that the task processing time $w(v_i)$ is a random variable, for which we are just given its probability distribution function. A similar assumption is made for communication times. Throughout the paper, task execution times and intertask communication times are supposed to be stochastically independent.

Some works have focused on stochastic task scheduling [16]-[21], [24], [25], [35]. Dong et al. proposed a mechanism to

- *K. Li, X. Tang, and K. Li are with the School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha 410082, China. E-mail: lkl510@263.net; tang_313@163.com; lik@newpaltz.edu.*
- *B. Veeravalli is with the Department of Electrical and Computer Engineering, National University of Singapore, 117576 Singapore. E-mail: elebv@nus.edu.sg.*
- *K. Li is also with the Department of Computer Science, State University of New York, New Paltz, NY 12561.*

Manuscript received 29 Dec. 2012; revised 21 July 2013; accepted 07 Oct. 2013; published online 18 Oct. 2013.
Recommended for acceptance by C.-Z. Xu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2012-12-0946.
Digital Object Identifier no. 10.1109/TC.2013.205

estimate the probability distribution of a task's execution time based on resource load, and a resource load based stochastic scheduling algorithm for grid environments was developed [25]. The weakness of the above technique is that computing the probability distribution of the makespan is inefficient and some solutions can never be selected even if they are not dominated (i.e., only solutions on the convex hull can be selected). There remain very challenging problems. Examples are, how to incorporate the expected values and variances of random computation and communication times into scheduling to improve the system performance, and how to deal with heterogeneity of processors in a cluster system. These issues are worth of further investigation, and there is still much room for improvement [10], [18], [24].

1.2 Our Contributions

In the present paper, we make new contribution to the problem of scheduling precedence constrained stochastic tasks of a parallel application on heterogeneous processors of a cluster system. We give a model of scheduling stochastic tasks on heterogeneous processors. We show that a lower bound for the expected makespan $E[C_{max}]$ is the makespan of scheduling deterministic tasks whose processing times and communication times are replaced by their expected values. We discuss effective ways to deal with various random variables in stochastic scheduling, such as the finish time of a processor, the finish time of a communication, the data ready time of a task, and the earliest execution start time and completion time of a task. In particular, we assume that all task processing times and intertask communication times have normal distributions, and we employ the well known Clark's equations [28], [31], [36]. In order to efficiently and effectively schedule precedence constrained stochastic tasks, we develop a stochastic dynamic level scheduling (SDLS) algorithm. We establish a stochastic task scheduling simulation platform and compare experimentally the SDLS scheduling algorithm with three existing heuristic scheduling algorithms, i.e., Rob-HEFT, SHEFT, and HEFT.

Our most significant contribution is to invent the concepts of stochastic bottom level and stochastic dynamic level (SDL), and to assign a task to a processor such that the SDL of the task on the processor is optimal according to a unique definition introduced in this paper. Our SDLS algorithm is able to effectively handle task dependency, time randomness, and processor heterogeneity simultaneously, leading to currently the best algorithm for scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. Our extensive experimental data on various kinds of task graphs demonstrate that our proposed algorithm can achieve reduced makespan, increased speedup, and reduced deviation of makespan.

2 Related Work

A popular representation of a parallel application is the *directed acyclic graph* (DAG), in which the nodes represent application tasks and their execution times, and the directed arcs or edges represent the execution dependencies as well as the amount of communication. It is widely known that the problem of finding an optimal schedule is NP-hard [5], even for very special cases. Therefore, a heuristic algorithm can be used to obtain a sub-optimal schedule rather than parsing all possible schedules. These heuristic algorithms can be broadly classified into the following three categories, namely, list scheduling algorithms [1], [2], [6], [7], clustering algorithms [3], [8], and duplication based algorithms [4].

Compared to algorithms from the other two categories, a list scheduling heuristic usually generates good quality schedules at a reasonable cost. The basic idea of list scheduling is to assign priorities to the tasks of a DAG and to place the tasks in a list arranged in descending order of priorities. A task with a higher priority is scheduled before a task with a lower priority and ties are broken using some method. To compute the priorities of the tasks, a DAG must be labeled with the processing times (weights) of the tasks and the communication times (weights) of the edges. Two frequently used attributes for assigning priority are the t_level (top level) and b_level (bottom level) [9]. The t_level of a task v_i is the length of a longest path (there can be more than one longest path) from an entry task to v_i (excluding v_i). Here, the length of a path is the sum of all the task and edge weights along the path. The b_level of a task v_i is the length of a longest path from v_i to an exit task. Examples of list scheduling algorithms are modified critical path (MCP) [7], [9], mobility directed (MD) [9], dynamic level scheduling (DLS) [2], [9], mapping heuristic (MH) [9], and dynamic critical path (DCP) [7].

However, all of the above works assume that task processing and communication times are deterministic and cannot be effectively applied to stochastic scheduling. Stochastic scheduling problems have been well studied in the scheduling theory literature over forty years. One of the earliest results in stochastic scheduling dated back to 1966, when Rothkopf gave a simple proof that $1|v_i \sim stoch|E[\sum w_i C_i]$ is optimally solved by greedily ordering tasks according to the *weighted shortest expected processing time* (WSEPT) policy [16]. This policy is so named because if all weights are equal, it becomes the *shortest expected processing time* (SEPT) policy, where we always execute the task having the shortest expected processing time. In the general case, Weiss analyzed the optimality gap of WSEPT, and proved that WSEPT is asymptotically optimal under mild assumptions on the input parameters of the problem [17]. A breakthrough occurred with the work of Möring et al. [10], who proved the general bounds for many stochastic completion time scheduling problems. Their method is based on LP relaxations of a deterministic problem, where an optimum solution to the LP yields a lower bound. Later, Megow et al. improved the results under a more general environment [11]. Scharbrodt et al. presented an average-case analysis for that problem [12].

Precedence constraints between tasks play a particularly important role in most real-world scheduling problems. Therefore, it would be of both theoretical and practical interest to incorporate such constraints into stochastic scheduling. Considering stochastic tasks with precedence constraints, such as the stochastic DAG model, Skutella and Uetz obtained a theoretical performance guarantee for weight execution time [18]. In most cases, the makespan is the key objective of stochastic scheduling with precedence constraints on heterogeneous systems. In scheduling theory, this problem is represented by the notation $Q|v_i \sim stoch, prec|E[C_{max}]$. The difficulty of this problem was impressively underlined by Hagstrom [19], who showed, among other things, that

computing the probability distribution of the makespan is an $\#P$-complete problem (Intuitively, a $\#P$-complete problem is to count the number of solutions of an NP-complete problem) even for a simple class of stochastic scheduling problems with precedence constraints.

Several authors have proposed solutions to estimate the distribution of the makespan. Canon and Jeannot looked upon the stochastic scheduling problem as a stable state of a system [20], [21]. They compared the robustness metrics (makespan standard deviation, makespan differential entropy, mean slack, probabilistic metric, lateness likelihood, makespan 0.99-quantile) and showed that a good metric is the *makespan standard deviation*. They also proposed a heuristic scheduling algorithm Rob-HEFT, which generates a set of solutions intended to have good performance for makespan and makespan standard deviation for stochastic applications. In our preliminary work [24], we proposed the *stochastic heterogeneous earliest finish time* (SHEFT) scheduling algorithm, which confirmed that the variance of a task's processing time is a key attribute and has much effect on performance in stochastic scheduling. However, the method of computing stochastic DAG path length is simple and should be effectively improved. Both Rob-HEFT and SHEFT algorithms will be compared with our proposed SDLS algorithm in this paper.

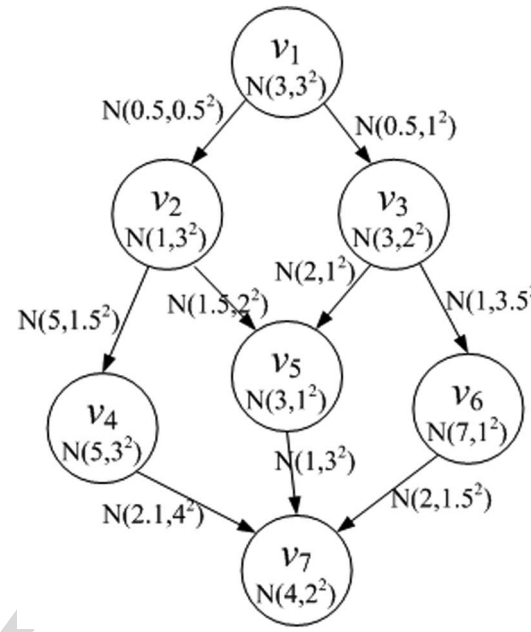

Fig. 1. An example of a stochastic parallel application.

3 A Stochastic Scheduling Model

3.1 Stochastic Parallel Applications

Generally, a stochastic parallel application with precedence constrained tasks is represented by a directed acyclic graph (DAG) $G = (V, E)$ [23], [24], where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n tasks that can be executed on any of the available processors, and $E \subseteq V \times V$ is a set of directed arcs or edges between the tasks to represent the dependencies. For example, edge $e_{i,j} \in E$ represents the precedence constraint such that task v_i should complete its execution before task v_j starts its execution. The weight $w(v_i)$ assigned to task v_i represents its processing time, and the weight $w(e_{i,j})$ assigned to edge $e_{i,j}$ represents its communication time. The set $\{v_j \in V : e_{j,i} \in E\}$ of all direct predecessors of v_i is denoted by pred (v_i), and the set $\{v_j \in V : e_{i,j} \in E\}$ of all direct successors of v_i is denoted by succ (v_i). A task may need input data from its predecessors. When all its input data are available, the task is triggered to execute, provided that there is an available processor or its designated processor is available. After its execution, a task generates its output data for its successors. A task $v_i \in V$ without predecessors, i.e., $\mathrm{pred}(v_i) = \emptyset$, is called an *entry task*. A task v_i without successors, i.e., $\mathrm{succ}(v_i) = \emptyset$, is called an *exit task*. Without loss of generality, we assume that a DAG has exactly one entry task v_{entry} and one exit task v_{exit}. If multiple entry tasks or exit tasks exist, they may be connected with zero-time weight edges to a single pseudo-entry task or a single pseudo-exit task that has zero-time weight.

In our stochastic setting, the processing and communication times are random and known only in advance as probability distributions. In this paper, we study the stochastic task scheduling problem with normal distributions of processing and communication times, and assume that the expected values and variances of the distributions are known. The assumption of normal distribution has been justified by many real applications, and also makes the analysis of many random variables analytically tractable. Such an approach has been adopted by many researchers [10], [24], [26]-[28]. Let $N(\mu, \sigma^2)$ denote the normal distribution with mean μ and variance σ^2. Then, the processing time of task v_i follows $w(v_i) \sim N(\mu_{v_i}, \sigma^2_{v_i})$, and the communication time of edge $e_{i,j}$ follows $w(e_{i,j}) \sim N(\mu_{e_{i,j}}, \sigma^2_{e_{i,j}})$, respectively.

Fig. 1 shows an example of a parallel application DAG with normal distributions of computation and communication times.

3.2 Heterogeneous Cluster Systems

In this study, we consider a heterogeneous cluster system which is modeled as a finite set P of m heterogeneous processors $p_1, p_2, \ldots, p_m$. Each processor consists of a single processing core, and each task must use a core exclusively. A weight $w(p_k)$ assigned to a processor p_k represents the processor's computation capacity, i.e., the relative speed compared to a standard reference processor. Thus, the execution time of task v_i on processor p_k is $w(v_i)/w(p_k)$. All processors are connected by a special network, such as a fast Ethernet. We assume that the communication time of two tasks v_i and v_j is $w(e_{i,j})$, if the two tasks are executed to two different processors, and the time is independent of the particular processors. However, the communication time between two tasks scheduled on the same processor is taken as zero [9], [23], [24]. The above computing system can be easily altered into a grid system by a little change and our scheduling model can be applied to grid systems.

3.3 A Motivational Example

To illustrate the complexity of scheduling stochastic tasks, which are known only their probability distributions in advance, and minimizing the makespan, we give a demonstration of the processing and communication times in Fig. 1. Table 1 provides the average-case processing times and the worst-case processing times of the tasks in Fig. 1, which are

TABLE 1
The Processing and Communication Times of Tasks in Fig. 1 for Deterministic Scheduling

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	$e_{1,2}$	$e_{1,3}$	$e_{2,4}$	$e_{2,5}$	$e_{3,5}$	$e_{3,6}$	$e_{4,7}$	$e_{5,7}$	$e_{6,7}$
average-case	3.0	1.0	3.0	5.0	3.0	7.0	4.0	0.50	0.5	5.0	1.5	2.0	1.0	2.1	1.0	2.0
worst-case	7.3	2.5	5.6	12.1	4.8	10.1	6.9	0.83	1.2	8.9	3.2	2.8	2.1	5.3	2.2	3.6

TABLE 2
The Execution Times of Tasks in Fig. 1 on a Cluster System

Number	v_1	v_2	v_3	v_4	v_5	v_6	v_7	$e_{1,2}$	$e_{1,3}$	$e_{2,4}$	$e_{2,5}$	$e_{3,5}$	$e_{3,6}$	$e_{4,7}$	$e_{5,7}$	$e_{6,7}$
1	4.5	0.70	2.3	12.0	3.9	5.6	2.90	1.50	0.20	7.3	0.90	2.60	0.71	3.22	1.43	1.21
2	3.3	2.10	2.9	8.1	3.4	9.1	6.40	0.81	0.30	3.2	2.82	1.80	1.32	4.13	1.92	2.16
3	3.5	1.30	2.1	8.3	3.1	6.7	4.90	0.32	0.88	1.3	1.20	2.10	1.40	0.90	0.30	0.90
4	3.2	0.89	3.6	3.5	3.3	9.3	3.12	0.56	0.80	6.7	1.89	2.73	1.90	2.30	0.90	2.80
5	1.9	0.34	4.2	5.8	4.7	4.7	4.32	0.62	0.94	3.4	1.11	2.28	0.70	3.20	1.10	1.60

used by deterministic scheduling algorithms. Table 2 gives the execution times of the tasks in Fig. 1, which are obtained from a cluster system based on Intel Xeon X5670 processors and Linux. From Table 2, we can conclude that a task's computation time or an edge's communication time can be far from its average-case and worst-case computation/communication time. Thus, the deterministic scheduling strategy is ineffective and we should use the stochastic scheduling approach to improve the performance of scheduling. The example will be continued and completed in later sections.

4 Preliminaries

4.1 Scheduling Attributes

To describe scheduling of an application DAG $G = (V, E)$ on a heterogeneous cluster system, the following notations are defined and used throughout the paper. $ST(v_i, p_k)$ denotes the *earliest execution start time* of task $v_i \in V$ on processor $p_k \in P$, and $ET(v_i, p_k) = w(v_i)/w(p_k)$ is the *execution time* of task v_i on processor p_k. As the computing capacities of processors in a heterogeneous cluster system are different from each other, the execution times, i.e., the $ET(v_i, p_k)$'s, are different too. The *earliest execution completion time* $C(v_i, p_k)$ of task v_i on processor p_k is calculated as

$$\begin{aligned} C(v_i, p_k) &= ST(v_i, p_k) + ET(v_i, p_k) \\ &= ST(v_i, p_k) + \frac{w(v_i)}{w(p_k)}. \end{aligned} \quad (1)$$

Notice that in this study, we assume that the processing time $w(v_i)$ is a random variable with normal distribution. Thus, the execution time $ET(v_i, p_k)$ and the earliest completion time $C(v_i, p_k)$ are random too.

The processor on which task v_i is executed is denoted by $prc(v_i)$. A *schedule* $s = (prc(v_1), prc(v_2), \ldots, prc(v_n))$ is a task assignment to the processors (equivalently, a processor allocation to the tasks). Furthermore, let

$$FT(p_k) = MAX_{prc(v_i)=p_k}\{C(v_i, p_k)\}$$

be the *finish time* of processor p_k. For a schedule to be feasible, the following condition must be satisfied by all tasks in the DAG [29], i.e., for any two tasks $v_i, v_j \in V$, we have

$$prc(v_i) = prc(v_j) = p_k \Rightarrow \begin{cases} C(v_i, p_k) \le ST(v_j, p_k), \\ or \\ C(v_j, p_k) \le ST(v_i, p_k). \end{cases}$$

The above condition essentially states that the execution of two tasks on the same processor cannot overlap.

The *data ready time* (DRT) of task v_i on processor p_k, represented by $DRT(v_i, p_k)$, is

$$DRT(v_i, p_k) = MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{C(e_{j,i})\}, \quad (2)$$

where $C(e_{j,i})$ is the *communication finish time* of edge $e_{j,i}$ calculated by

$$C(e_{j,i}) = C(v_j, p_{prc(v_j)}) + w(e_{j,i}). \quad (3)$$

If $pred(v_i) = \emptyset$, i.e., v_i is an entry task, then we have $DRT(v_i, p_k) = 0$, for all $p_k \in P$. The start time $ST(v_i, p_k)$ of task v_i on processor p_k is constrained by processor p_k's finish time $FT(p_k)$ and the entering edges of v_i as follows,

$$ST(v_i, p_k) = MAX\{DRT(v_i, p_k), FT(p_k)\}, \quad (4)$$

for all $v_i \in V$ and $p_k \in P$.

The *makespan* is $C_{max} = C(v_{exit}, p_{prc(v_{exit})})$, assuming that $ST(v_{entry}, p_{prc(v_{entry})}) = 0$. In solving the $Q|v_i \sim stoch, prec\ |E[C_{max}]$ problem, one of the objectives is to minimize the average makespan $E[C(v_{exit}, p_{prc(v_{exit})})]$.

4.2 Manipulation of Normal Random Variables

Since the processing time of a task and the communication time of an edge in a stochastic parallel application are given as random variables, many quantities in stochastic scheduling, such as the length of a path from a task to the exit task, are also random variables, which differ from the deterministic ones. Before presenting our stochastic dynamic level scheduling

(SDLS) algorithm for the $Q|v_i \sim stoch, prec|E[C_{max}]$ problem, we would like to explain how to calculate the probability distributions of various random variables involved in our stochastic scheduling.

There are several basic operations on normal random variables which are employed throughout this paper. First, a normal random variable $X \sim N(\mu, \sigma^2)$ can be scaled by a constant factor c, e.g., the random variable $w(v_i)$ is divided by a constant $w(p_k)$ in Eq. (1). It is clear that $cX \sim N(c\mu, (|c|\sigma)^2)$.

Second, we encounter summations of normal random variables. An important fact about normal random variables is that if X_i is normally distributed with expected value μ_i and variance σ_i^2, where $i = 1, 2, \ldots, n$, then $X = \Sigma_i^n X_i$ is also normally distributed with parameters $\Sigma_i^n \mu_i$ and $\Sigma_i^n \sigma_i^2$. In other words,

$$X \sim N(\Sigma_i^n \mu_i, \Sigma_i^n \sigma_i^2). \tag{5}$$

This attribute can be applied to calculate the distribution of the length of a path consisting of tasks and edges in a DAG, such as Fig. 2a. Here, the length distribution of path $(v_1 \rightarrow e_{1,2} \rightarrow v_2)$ in Fig. 2a is $N(\mu_1 + \mu_{1,2} + \mu_2, \sigma_1^2 + \sigma_{1,2}^2 + \sigma_2^2)$. We also notice that $X - Y = X + (-Y)$, i.e., the difference of two normal random variables is also a normal random variable. This property will be used in Eq. (14).

Third, we need to find the maximum of several random variables, such as $DRT(v_i, p_k)$ in Eq. (2) and the MAX operator in Eq. (4). Unfortunately, the maximum value of a set of normal random variables is no longer a normal random variable. However, in a pioneering work [28], [31], [36], Clark developed a method to recursively estimate the expected value and variance of the greatest value of a finite set of random variables that are normally distributed. The normal distribution based on the obtained expected value and variance is close to the actual distribution of the random variable defined by the MAX operator. Let us consider the tasks in Fig. 2b, where task v_n has $n-1$ predecessors. The data ready time of task v_n can be expressed as

$$\begin{aligned} DRT(v_n) &= MAX\{C(e_{1,n}), C(e_{2,n}), \ldots, C(e_{n-1,n})\} \\ &= MAX\{MAX\{C(e_{1,n}), C(e_{2,n})\}, \ldots, C(e_{n-1,n})\}. \end{aligned} \tag{6}$$

The above equation implies that the distribution of $DRT(v_n)$ can be calculated recursively by using Clark's method elaborated as follows.

As the communication times are independent and normally distributed, the correlation coefficient between any pair of them is zero. That is,

$$\rho_{i,j} = \rho(C(e_{i,n}), C(e_{j,n})) = 0, \quad \forall\, i, j = 1, 2, \ldots, n-1, \quad and\ i \neq j.$$

(*Note*: Strictly speaking, although the above condition holds for Fig. 2b, it is not true when Fig. 2b is part of a DAG. The reason is that the task completion times of the predecessors are not independent, although the edge communication times are independent. However, since we are only interested in getting an approximate probability distribution for the MAX operator, we still adopt this assumption, so that Clark's equations can be applied easily.) Clark's equations

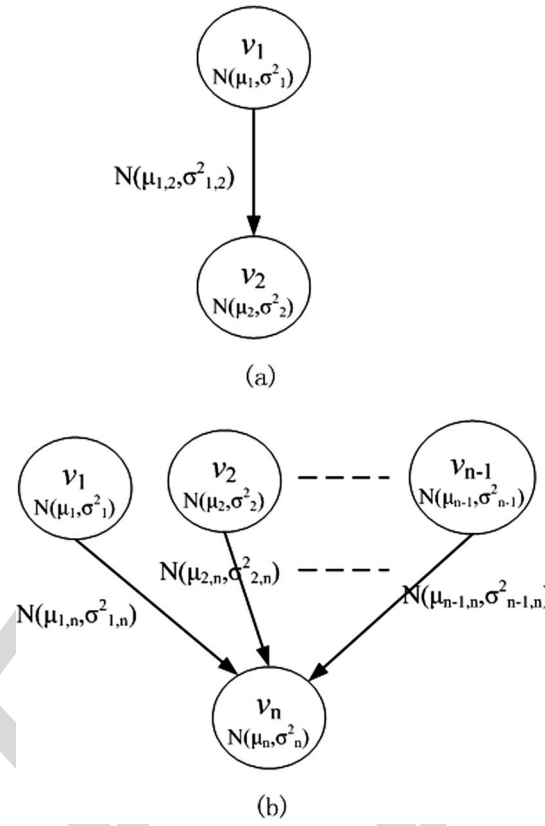

Fig. 2. Example random variables in stochastic scheduling.

to compute the expected value and variance of $MAX\{C(e_{1,n}), C(e_{2,n})\}$ with $\rho_{1,2} = 0$ are as follows. The expectation of $MAX\ \{C(e_{1,n}), C(e_{2,n})\}$ is given by Clark's first equation. That is,

$$\begin{aligned} E[MAX\{C(e_{1,n}), C(e_{2,n})\}] &= E[C(e_{1,n})]\Phi(\xi_{1,2}) \\ &\quad + E[C(e_{2,n})]\Phi(-\xi_{1,2}) + \varepsilon_{1,2}\psi(\xi_{1,2}) \\ &= E[C(e_{2,n})] + (E[C(e_{1,n})] \\ &\quad - E[C(e_{2,n})])\Phi(\xi_{1,2}) + \varepsilon_{1,2}\psi(\xi_{1,2}), \end{aligned} \tag{7}$$

where,

$$\begin{aligned} \varepsilon_{1,2} &= \sqrt{Var[C(e_{1,n})] + Var[C(e_{2,n})] - 2\rho_{1,2}\chi_{1,2}} \\ &= \sqrt{Var[C(e_{1,n})] + Var[C(e_{2,n})]} \end{aligned}$$

(since $\rho_{1,2} = 0$,) and

$$\xi_{1,2} = \frac{E[C(e_{1,n})] - E[C(e_{2,n})]}{\varepsilon_{1,2}},$$

and $\chi_{1,2} = \sqrt{Var[C(e_{1,n})]Var[C(e_{2,n})]}$, and $\psi(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}$, and

$$\Phi(x) = \int_{-\infty}^{x} \psi(t)dt = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2} dt.$$

The variance of $MAX\{C(e_{1,n}), C(e_{2,n})\}$ is given by Clark's second equation. That is,

$$Var[MAX\{C(e_{1,n}), C(e_{2,n})\}] = (E^2[C(e_{1,n})] + Var[C(e_{1,n})])\Phi(\xi_{1,2}) + (E^2[C(e_{2,n})] + Var[C(e_{2,n})]) \times \Phi(-\xi_{1,2}) + (E[C(e_{1,n})] + E[C(e_{2,n})])\varepsilon_{1,2}\psi(\xi_{1,2}) - E^2[MAX\{C(e_{1,n}), C(e_{2,n})\}]. \quad (8)$$

Now, let us consider

$$MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\} = MAX\{MAX\{C(e_{1,n}), C(e_{2,n})\}, C(e_{3,n})\}.$$

The correlation coefficient $\rho_{1,2,3}$ becomes the correlation coefficient between $MAX\{C(e_{1,n}), C(e_{2,n})\}$ and $C(e_{3,n})$. It is clear that $\rho_{1,2,3} = 0$. The expected value of $MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\}$ is as follows:

$$E[MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\}] = E[MAX\{C(e_{1,n}), C(e_{2,n})\}]\Phi(\xi_{1,2,3}) + E[C(e_{3,n})]\Phi(-\xi_{1,2,3}) + \varepsilon_{1,2,3}\psi(\xi_{1,2,3}),$$

where,

$$\varepsilon_{1,2,3} = \sqrt{Var[MAX\{C(e_{1,n}), C(e_{2,n})\}] + Var[C(e_{3,n})]},$$

and

$$\xi_{1,2,3} = \frac{E[MAX\{C(e_{1,n}), C(e_{2,n})\}] - E[C(e_{3,n})]}{\varepsilon_{1,2,3}}.$$

The variance of $MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\}$ can be computed as

$$Var[MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\}] = (E^2[MAX\{C(e_{1,n}), C(e_{2,n})\}] + Var[MAX\{C(e_{1,n}), C(e_{2,n})\}])\Phi(\xi_{1,2,3}) + (E^2[C(e_{3,n})] + Var[C(e_{3,n})])\Phi(-\xi_{1,2,3}) + (E[MAX\{C(e_{1,n}), C(e_{2,n})\}] + E[C(e_{3,n})])\varepsilon_{1,2,3}\psi(\xi_{1,2,3}) - E^2[MAX\{C(e_{1,n}), C(e_{2,n}), C(e_{3,n})\}].$$

In a similar way, Clark's equations can be used recursively for $(n-3)$ more steps to determine the expected value and variance of $DRT(v_n)$.

5 A Lower Bound on the Expected Makespan

For the scheduling problem $Q|v_i \sim stoch, prec|E[C_{max}]$, a lower bound for the expected makespan can be obtained by replacing all processing and communication times by their expected values and turning the stochastic model into a deterministic model. Let C_i^s denote the completion time of task v_i under a schedule s. The makespan under s is $C_{max}^s = C_{exit}^s$. We define a deterministic DAG $\hat{G}(V, E)$ in the same way as $G(V, E)$, but replacing each random variable by its expected value. Let $\hat{C}_i^s$ be the completion time of task v_i in $\hat{G}(V, E)$ and $\hat{C}_{max}^s = \hat{C}_{exit}^s$ be the makespan under schedule s.

The following theorem gives a lower bound for the expected makespan.

Theorem 1. *For the problem $Q|v_i \sim stoch, prec|E[C_{max}]$ with random processing and communication times, we have $E[C_i^s] \geq \hat{C}_i^s$ for all $v_1, v_2, \ldots, v_n$. Consequently, the expected makespan satisfies the following inequality:*

$$E[C_{max}^s] \geq \hat{C}_{max}^s, \quad (9)$$

for any schedule s.

Proof. We prove $E[C_i^s] \geq \hat{C}_i^s$ for all v_i by a *structural induction* on a DAG, i.e., the statement is true for v_i as long as it holds for all its predecessors. First, for the *induction basis*, we notice that

$$E[C^s(v_{entry})] = \frac{E[w(v_{entry})]}{w(p_{prc(v_{entry})})} = \hat{C}^s(v_{entry}),$$

that is, the statement if true for a task without any predecessor. Next, for the *induction step*, we consider any task v_i with predecessors. Assume that under schedule s, task v_i is processed by processor p_k. Hence, by Eq. (1), we have

$$E[C_i^s] = E\left[ST^s(v_i, p_k) + \frac{w(v_i)}{w(p_k)}\right],$$

where the start time can be obtained by Eqs. (2) and (4),

$$ST^s(v_i, p_k) = MAX\{DRT^s(v_i, p_k), FT^s(p_k)\} = MAX\{MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{C^s(e_{j,i})\}, FT^s(p_k)\}.$$

Thus, we get

$$E[C_i^s] = E\left[MAX\{MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{C^s(e_{j,i})\}, FT^s(p_k)\} + \frac{w(v_i)}{w(p_k)}\right].$$

Since the above MAX function is convex [30], by applying Jensen's inequality, we obtain

$$E[C_i^s] \geq MAX\{MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{E[C^s(e_{j,i})]\}, E[FT^s(p_k)]\} + \frac{E[w(v_i)]}{w(p_k)}.$$

The communication finish time $C^s(e_{j,i})$ is the sum of the task completion time C_j^s and the communication time $w(e_{j,i})$ of edge $e_{j,i}$. Since the finish time $FT^s(p_k)$ of processor p_k is determined by the expected processing times of its assigned tasks, $E[FT^s(p_k)]$ is equal to $\hat{FT}^s(p_k)$ of the deterministic DAG $\hat{G}(V, E)$ under schedule s. The above inequality becomes

$$\begin{aligned} E[C_i^s] &\geq MAX\{MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{E[C_j^s] + E[w(e_{j,i})]\}, \hat{FT}^s(p_k)\} + \frac{E[w(v_i)]}{w(p_k)} \\ &\geq MAX\{MAX_{v_j \in pred(v_i), prc(v_j) \neq p_k}\{\hat{C}_j^s + E[w(e_{j,i})]\}, \hat{FT}^s(p_k)\} + \frac{E[w(v_i)]}{w(p_k)} \\ &= \hat{C}_i^s, \end{aligned}$$

where the last inequality is from the *induction hypothesis* that the inequality $E[C_j^s] \geq \hat{C}_j^s$ holds for all task v_i's predecessors v_j. Finally, we notice that $E[C_{max}^s] = E[C_{exit}^s] \geq \hat{C}_{exit}^s = \hat{C}_{max}^s$. This completes the proof of the theorem. □

The above lower bound theorem can be used to evaluate the quality of a schedule s in solving the $Q|v_i \sim stoch, prec|E[C_{max}]$ problem.

6 A Stochastic Scheduling Algorithm

6.1 Stochastic Bottom Level

In a heterogeneous cluster system, the different computation times of the same task on different processors present us with a problem, that is, the priorities computed using the task computation times on one particular processor may differ from the priorities computed using the task computation times on another processor. To overcome this problem, some scheduling algorithms set the task computation times to their median values, as in the DLS algorithm [2]. Some set to their average values, as in the HEFT algorithm [23]. For heterogeneous processors, the impacts of different computation capacity assignment methods are investigated in [32], where the authors compared different methods such as average value, median value, best value, and worst value. Results show that the average value is the most suitable method for heterogeneous cluster systems. Here, we use the average computation capacity, which is defined as

$$\overline{w(p)} = \frac{1}{m}\sum_{k=1}^{m} w(p_k). \tag{10}$$

In deterministic scheduling algorithms, the key attributes such as top level t_level and bottom level b_level can be easily computed by methods proposed in [9]. However, for stochastic scheduling algorithms, it is difficulty to compute the above attributes, because all processing times of tasks and communication times among tasks are random. Here, we propose an algorithm to calculate the distribution of b_level, which we call *stochastic bottom level* represented by sb_level. The sb_level of a task v_x is the random length of a longest path from v_x to the exit task, and is recursively defined as

$$sb\_level(v_x) = MAX_{v_i \in succ(v_x)}\{w(e_{x,i}) + sb\_level(v_i)\} + \frac{w(v_x)}{\overline{w(p)}}. \tag{11}$$

For the exit task v_{exit}, its sb_level is

$$sb\_level(v_{exit}) = \frac{w(v_{exit})}{\overline{w(p)}}. \tag{12}$$

Algorithm 1 is developed for computing the sb_level of all tasks in a parallel application.

Algorithm 1: Computing the sb_level

Input: A task DAG of a parallel application.

Output: The tasks' stochastic bottom levels.

1 Construct a list of tasks in a reversed topological order and call it *RevTopList*;

2 Compute $sb\_level(v_{exit})$ by using Eq. (12) and remove task v_{exit} from *RevTopList*;

3 **while** the *RevTopList* is not empty

4 Remove the first task v_x from *RevTopList*;

5 Get a child v_i of v_x and use Eq. (5) to compute the expected value and variance of $sb\_level(v_x) = w(e_{x,i}) + sb\_level(v_i)$;

6 **for** each other child v_i of v_x **do**

7 Use Eqs. (7) and (8) to compute the expected value and variance of $sb\_level(v_x) = MAX\{sb\_level(v_x), w(e_{x,i}) + sb\_level(v_i)\}$

8 **end**

9 Use the expected value and variance of $sb\_level(v_x)$ to construct an approximate normal distribution of $sb\_level(v_x)$;

10 Use Eq. (5) to compute $sb\_level(v_x) = sb\_level(v_x) + w(v_x)/\overline{w(p)}$;

11 **end**

As an illustration, Table 3 presents the sb_level values obtained by Algorithm 1 for the sample DAG in Fig. 1 on a heterogeneous system with average computation capacity $\overline{w(p)} = 1$.

6.2 Stochastic Dynamic Level Scheduling

In this section, we present our Stochastic Dynamic Level Scheduling (SDLS) algorithm, which is derived from DLS [2], [9], and is based on *stochastic dynamic level* (SDL) to be defined below.

For heterogeneous systems, the computation capacities of processors are different. To account for the varying computation capacity, we define

$$\Delta(v_i, p_k) = \frac{w(v_i)}{\overline{w(p)}} - \frac{w(v_i)}{w(p_k)}. \tag{13}$$

A large positive $\Delta(v_i, p_k)$ indicates that processor p_k is faster than most processors, while a large negative $\Delta(v_i, p_k)$ indicates the opposite. The $SDL(v_i, p_k)$ of task v_i on processor p_k is defined as task v_i's stochastic bottom level $sb\_level(v_i)$ subtracted by task v_i's earliest execution start time $ST(v_i, p_k)$ on p_k, and added by $\Delta(v_i, p_k)$, as given by the following equation:

$$SDL(v_i, p_k) = sb\_level(v_i) - ST(v_i, p_k) + \Delta(v_i, p_k). \tag{14}$$

The SDL has a straightforward interpretation. When considering prospective tasks for scheduling, a task v_i with high

TABLE 3
The sb_level Values of Tasks in Fig. 1

v_1	v_2	v_3	v_4	v_5	v_6	v_7
$N(24.20, 26.80)$	$N(17.76, 33.20)$	$N(17.86, 18.00)$	$N(11.10, 29.00)$	$N(8.00, 14.00)$	$N(13.00, 7.25)$	$N(4.00, 4.00)$

stochastic bottom level $sb\_level(v_i)$ is desirable, because it indicates a high priority for execution. When comparing candidate processors for allocation, a processor p_k with later starting time $ST(v_i, p_k)$ is undesirable; however, a processor p_k with large $\Delta(v_i, p_k)$ is desirable.

Algorithm 2: The SDLS Algorithm

Input: A task DAG of a parallel application.

Output: A schedule $s = (prc(v_1), \ldots, prc(v_n))$.

1 Calculate sb_level of each task using Algorithm 1;
2 Push the entry task v_{entry} into the ready task pool;
3 **While** *the ready task pool is not empty* **do**
4 **for** *each task* v_i *in the ready task pool* **do**
5 **for** *each processor* p_k **do**
6 Use Eq. (14) to compute $SDL(v_i, p_k)$
7 **end**
8 **end**
9 Find the optimal task-processor pair (v_i, p_k) whose $SDL(v_i, p_k)$ is stochastically greater than the SDL of all other task-processor pairs;
10 Remove task v_i from the ready task pool;
11 Assign task v_i to processor p_k, i.e., $prc(v_i) = p_k$;
12 Push the unconstrained child tasks of v_i into the ready task pool;
13 Update the earliest execution start times of tasks on processors;
14 **end**

The SDLS algorithm is formally described in Algorithm 2. Similar to the DLS algorithm, in each repetition of the while loop, the SDLS algorithm uses Eq. (14) to compute the SDL of all ready tasks on all processors in a heterogeneous cluster system. Then, the task-processor pair (v_i, p_k) which gives the maximum $SDL(v_i, p_k)$ is selected and task v_i is scheduled on processor p_k. However, the difference between SDLS and DLS is how to select the task-processor pair with the maximum SDL, because the SDL's are random variables with normal distributions. Here, we define an operator $\prec$ to get the maximum SDL. Let $F_X(x)$ denote the cumulative density function of a random variable X. Assume that X_1 and X_2 are two random variables with normal distributions, and x_1 is the point such that $F_{X_1}(x_1) = 0.9$, and x_2 is the point such that $F_{X_2}(x_2) = 0.9$. We say that X_1 is *stochastically less than* X_2, or X_2 is *stochastically greater than* X_1, denoted by $X_1 \prec X_2$, if and only if $x_1 < x_2$. Algorithm SDLS finds the optimal task-processor pair (v_i, p_k) whose $SDL(v_i, p_k)$ is stochastically greater than the SDL of all other task-processor pairs. Fig. 3 shows two cumulative density functions and also demonstrates that the random variable X_2 is stochastically greater than X_1. The operator $\prec$ can get the greater SDL in a stochastic sense. The probability 0.9 is high enough to yield good scheduling performance.

As an illustration, consider the stochastic tasks in the DAG shown in Fig. 1 and a heterogeneous cluster system with 3 processors p_0, p_1, and p_2, where $w(p_0) = 0.7$, $w(p_1) = 1.4$, and $w(p_2) = 0.9$. The schedules generated by the classical deterministic list scheduling algorithm DLS [2], [9], which uses the tasks' mean processing and communication times, and our proposed algorithm SDLS, are shown in Figs. 4a and 4b respectively. The processing times and communication times of Fig. 4 are obtained from their real values on processors, such as, task v_1's processing time is 2.27 on processor p_1 and task v_4's processing time is 5.45 on processor p_2. The schedule generated by the SDLS algorithm has makespan 14.46, which is shorter than the makespan 15.33 of the schedule generated by the DLS algorithm.

7 Performance Evaluation

In this section, we compare the performance of the SDLS algorithm with three existing scheduling algorithms, i.e., Rob-HEFT [21], SHEFT [24], and HEFT [23]. Algorithms Rob-HEFT and SHEFT are heuristic algorithms for scheduling stochastic tasks. The main idea of Rob-HEFT is to choose the processor that leads to the desired trade-off between makespan and standard deviation. The SHEFT algorithm simply transforms the expected value and variance of stochastic task execution time into deterministic time. Both Rob-HEFT and SHEFT attempt to get the approximation of task execution

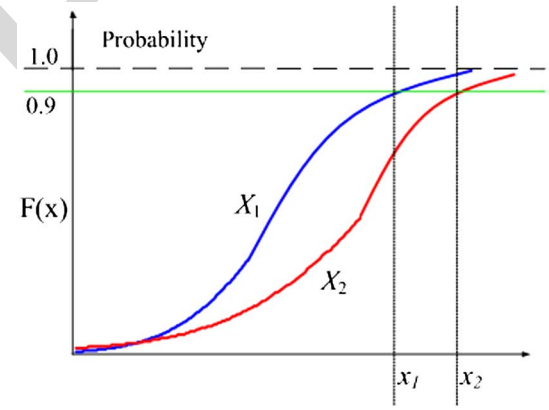

Fig. 3. Illustration of the operator $\prec$.

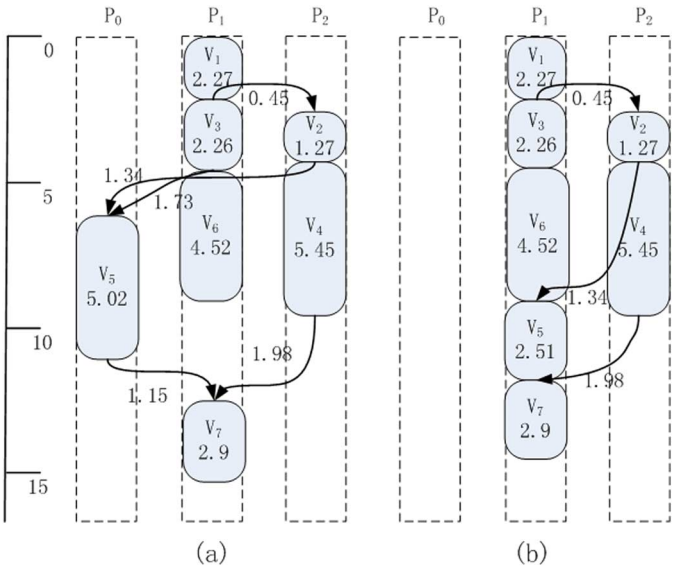

Fig. 4. Scheduling of the stochastic tasks in the DAG of Fig. 1 by using (a) DLS (makespan = 15.33); (b) SDLS (makespan = 14.46).

times, with precision lower than SDLS. The HEFT algorithm is a deterministic one and not very suitable for stochastic task scheduling.

We have built a simulation environment for cluster systems with 16 processors, with their computation capacities in the range from 2,000 MIPS to 3,000 MIPS. Stochastic parallel application graphs are randomly generated by varying parameters such as the size of a DAG, the height of a DAG, the link density, the minimum and maximum expected values $(T\mu_{min}, T\mu_{max})$ and variances $(T\sigma_{min}, T\sigma_{max})$ of task processing times, the minimum and maximum expected values $(E\mu_{min}, E\mu_{max})$ and variances $(E\sigma_{min}, E\sigma_{max})$ of communication times among tasks. The comparison is intended not only to present quantitative results, but also to qualitatively analyze the results and to explain the reasons, for better understanding of the stochastic scheduling problem.

7.1 Performance Metrics

The comparison of the algorithms are based on the following three performance metrics.

- *Makespan*—The makespan (or schedule length) is defined as the completion time of the exit task v_{exit}.
- *Speedup*—The speedup is computed by dividing the sequential execution time (i.e., the cumulative execution time) by the parallel execution time (i.e., the makespan of the output schedule)[23], [24], [37] as shown in Eq. (15):

$$Speedup = \frac{\sum_{v_i \in V} ET(v_i)}{makespan}, \tag{15}$$

  where $ET(v_i)$ is the execution time of task v_i. The sequential execution time is computed by assigning all stochastic tasks to a single processor that minimizes the cumulative of the computation times. If the sum of the computational times is maximized, it results in higher speedup, but ends up with the same ranking of the scheduling algorithms.
- *Makespan standard deviation*—Intuitively, the standard deviation of the makespan tells how narrow the makespan distribution is [21]. The narrower the distribution, the smaller the standard deviation. This metric is examined in this paper, because when we are given two schedules, the one with smaller makespan standard deviation is more likely to have stable performance in a cluster system.

7.2 Randomly Generated Application Graphs

In our experiments, we considered randomly generated application graphs, whose task processing times and edge communication times are assumed to be normally distributed [10], [24], [26]-[28]. Our simulation-based framework allows assigning sets of values to the parameters used by the random graph generator. This framework first executes the random graph generator program to construct application DAGs. Then, it executes the scheduling algorithms to generate output schedules. Finally, it computes the performance metrics based on the schedules. For the generation of random graphs, which are commonly used to compare scheduling algorithms [2], [6], [7], [9], five fundamental characteristics of a DAG are considered.

- Size n of a DAG: The number of stochastic tasks in an application DAG is n.
- Height h of a DAG: The n tasks are randomly partitioned into h levels.
- Link density β: The probability $q_{i,j}$ that there is a directed edge from a task of level i to a task of level j is

$$q_{i,j} = \frac{\beta}{j-i}, \tag{16}$$

  where $j > i$, and $i, j \in \{1, 2, \ldots, h\}$.
- The minimum and maximum expected values $(T\mu_{min}, T\mu_{max})$ and variances $(T\sigma_{min}, T\sigma_{max})$ of task processing times: The expected value and variance of each task processing time on every processor are uniform random variables in the intervals $[T\mu_{min}, T\mu_{max}]$ and $[T\sigma_{min}, T\sigma_{max}]$ respectively.
- The minimum and maximum expected values $(E\mu_{min}, E\mu_{max})$ and variances $(E\sigma_{min}, E\sigma_{max})$ of communication times among tasks: The expected value and variance of the communication time on each edge are uniform random variables in the intervals $[E\mu_{min}, E\mu_{max}]$ and $[E\sigma_{min}, E\sigma_{max}]$ respectively.

In our simulation experiments, graphs are generated for different combinations of the above parameters with number of tasks ranging between 50 and 300. Every possible edge is created with an appropriate link density, which is calculated based on the average number of edges per task node. Every set of the above parameters are used to generate several random graphs in order to avoid scattering effects. The results presented below are the average of the results obtained from these graphs. To make the comparison fair, all scheduling algorithms (SDLS, Rob-HEFT, SHEFT, and HEFT) are given the same parameters.

7.3 Randomly Generated DAG Experimental Results

For the first set of simulation studies, we compare and analyze the performance metrics by changing the number of processors from 4 to 16 in steps of 2. The results are shown in Figs. 5, 6, and 7, where each data point is the average of the data

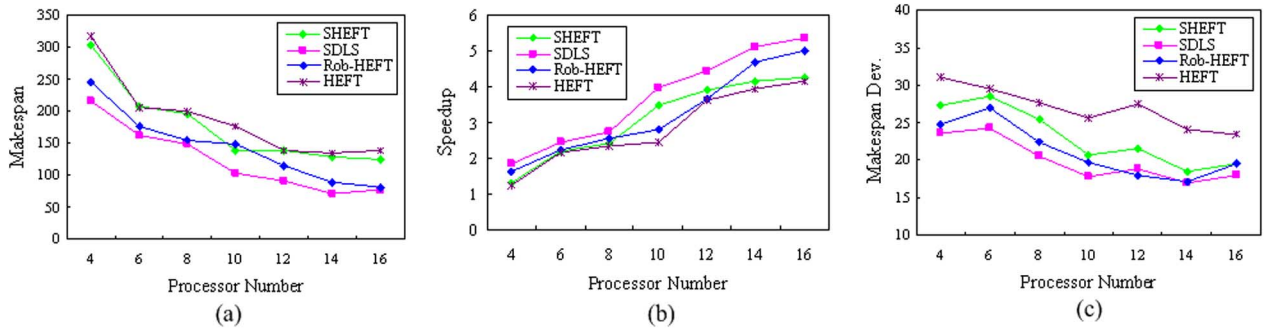

Fig. 5. Performance data for 100 tasks: (a) Makespan; (b) Speedup; (c) Makespan standard deviation.

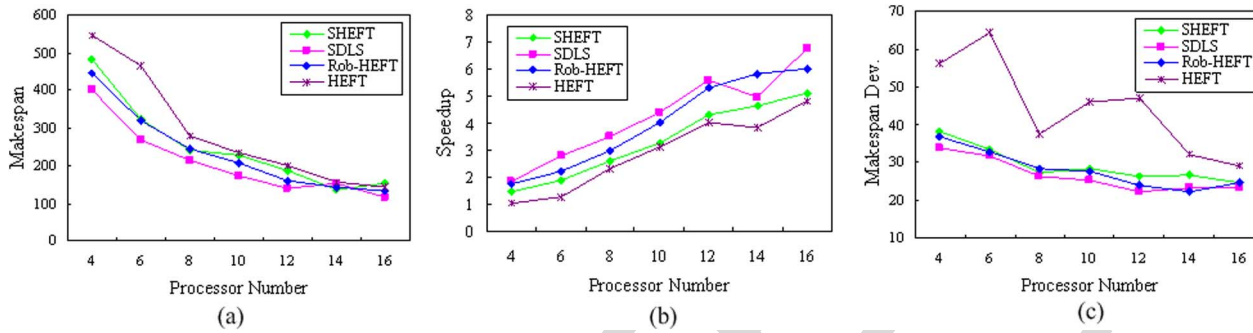

Fig. 6. Performance data for 200 tasks: (a) Makespan; (b) Speedup; (c) Makespan standard deviation.

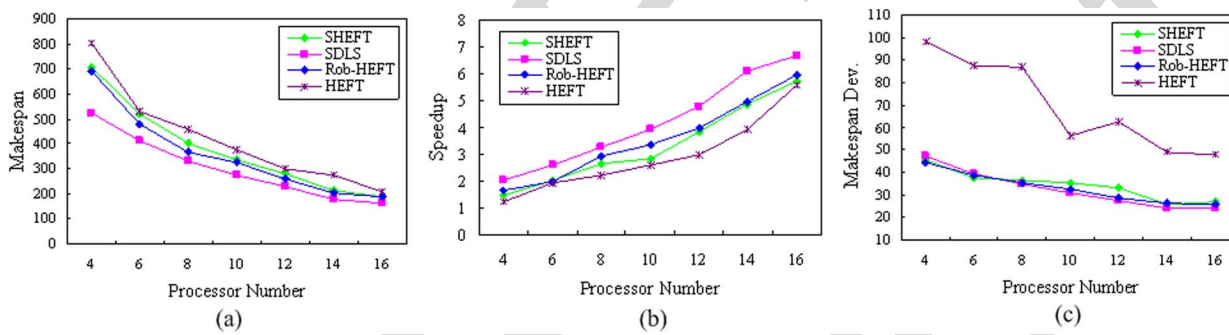

Fig. 7. Performance data for 300 tasks: (a) Makespan; (b) Speedup; (c) Makespan standard deviation.

obtained from 100 experiments. Fig. 5 shows the simulation results of the four scheduling strategies on cluster systems for stochastic application DAGs with 100 tasks. We observe from Fig. 5 that the stochastic scheduling algorithms SDLS, Rob-HEFT, and SHEFT, which take the random variable attributes such as expected value and variance into account, are better than the deterministic scheduling algorithm HEFT in terms of makespan, speedup, and makespan standard deviation. We also observe that our proposed algorithm SDLS significantly outperforms Rob-HEFT by 14.07%, SHEFT by 30.03%, HEFT by 33.85%, respectively, in term of the average makespan. For the average speedup, SDLS outperforms Rob-HEFT by 15.0%, SHEFT by 19.1%, HEFT by 30.2%, respectively. Furthermore, we conclude from Fig. 5c that SDLS has more stable performance than the other three algorithms, since the makespan standard deviation of SDLS is less than those of Rob-HEFT, SHEFT, and HEFT. This is due to the fact that SDLS considers the expected values and variances of task processing times and edge communication times, and produces better schedules for the stochastic scheduling problem $Q|v_i \sim stoch, prec|E[C_{max}]$. However, the other scheduling algorithms only consider the expected values of task processing times and edge communication times, and are less suitable for stochastic task scheduling. Finally, we would like to mention that as the processor number increases, the makespan and makespan standard deviation of the four scheduling strategies decrease, and the speedup of them increase.

From Fig. 5, we observe that SDLS is better than the other two stochastic scheduling algorithms Rob-HEFT and SHEFT. This is mainly due to two advanced techniques employed by our algorithm. First, the SDLS algorithm is able to handle normally distributed task processing times and edge communication times more effectively by using the techniques developed in this paper, especially Clark's equations and our method of comparing random variables. Second, the SDLS algorithm uses two key techniques, i.e., stochastic bottom level sb_level and stochastic dynamic level SDL, to improve the quality of scheduling. However, the

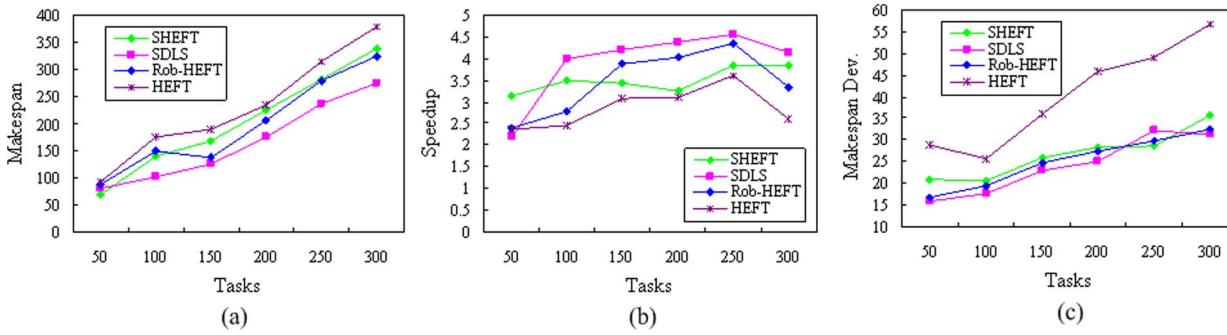

Fig. 8. Performance data for 10 processors: (a) Makespan; (b) Speedup; (c) Makespan standard deviation.

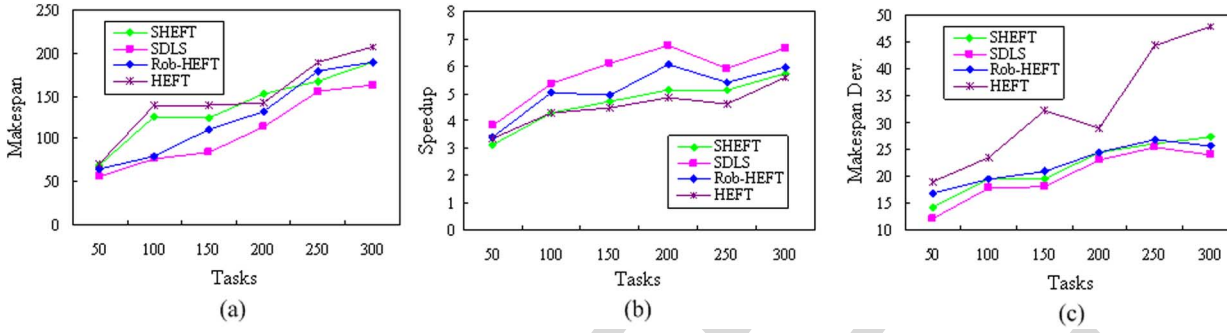

Fig. 9. Performance data for 16 processors: (a) Makespan; (b) Speedup; (c) Makespan standard deviation.

Rob-HEFT algorithm which uses a criteria space (makespan and standard deviation), and the SHEFT algorithm which uses the approximate weights of random variables, are not able to predict the distributions of various random scheduling attributes, and have lower quality of scheduling stochastic tasks.

The improvements of SDLS over Rob-HEFT, SHEFT, and HEFT could also be concluded from Figs. 6 and 7. Fig. 6 shows the simulation results for stochastic parallel application DAGs with 200 tasks. The results demonstrate that SDLS outperforms Rob-HEFT by 11.50%, SHEFT by 16.40%, HEFT by 27.73%, respectively, in term of the average makespan. For the average speedup, SDLS outperforms Rob-HEFT by 6.4%, SHEFT by 28.3%, HEFT by 31.7%, respectively. We also observe from Fig. 6c that SDLS outperforms Rob-HEFT by 5.56%, SHEFT by 10.22%, HEFT by 40.50%, respectively, in term of makespan standard deviation. Fig. 7 shows the simulation results for stochastic parallel application DAGs with 300 tasks. Again, SDLS performs better than Rob-HEFT, SHEFT, and HEFT in terms of makespan, speedup, and makespan standard deviation. From these experimental results, we can conclude that the integration of the variances of random variables into consideration, such as what SDLS does, has significant impact on the performance of a stochastic scheduling algorithm for the problem $Q|v_i \sim stoch, prec|E[C_{max}]$, and is likely to yield better scheduling performance.

In the second set of simulation experiments, we compare and analyze the performance metrics by changing the DAG size from 50 to 300 with step 50. The results reported in Fig. 8 for 10 processors and Fig. 9 for 16 processors reveal that the SDLS algorithm outperforms the Rob-HEFT, SHEFT, HEFT algorithms, in terms of makespan, speedup, and makespan standard deviation. As the stochastic parallel application DAG size increases, the makespan and makespan standard deviation increase too. However, the speedup does not always increase as the DAG size increases, with the best speedup at roughly 200 tasks. The above simulation results also show the fact that the deterministic scheduling algorithms such as HEFT and DLS are not suitable for the stochastic scheduling problem $Q|v_i \sim stoch, prec|E[C_{max}]$.

7.4 Special DAG Experimental Results

In this set of simulation experiments, we examine the performance of these four algorithms in scheduling some special DAGs. One special DAG in Fig. 10a has 50 tasks of width no more than 3, which represents a parallel application with very

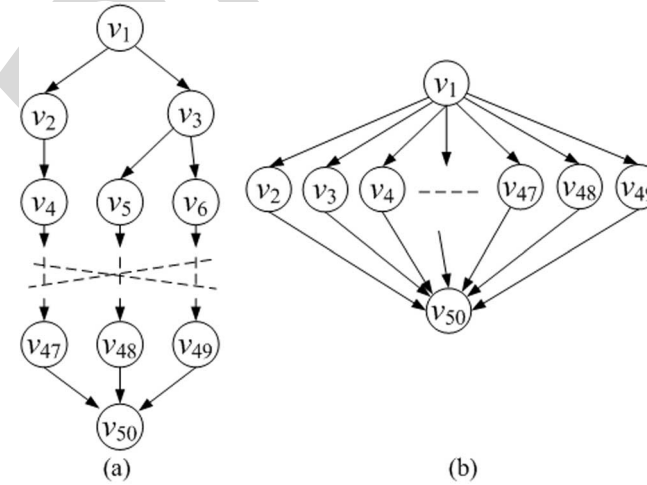

Fig. 10. Examples of special DAGs: (a) A low parallelism degree application; (b) A high parallelism degree application.

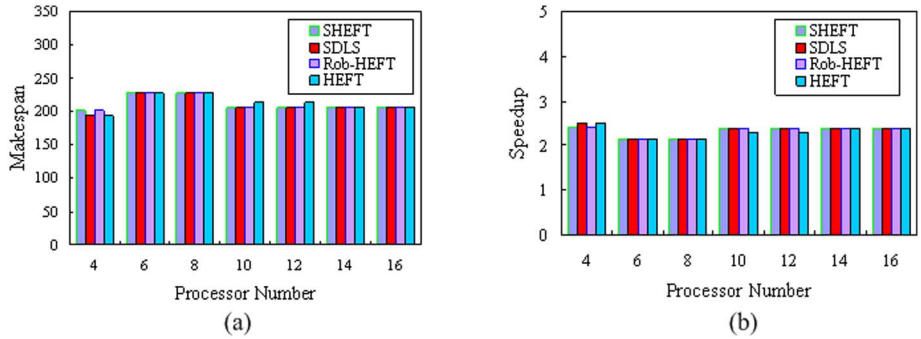

Fig. 11. Experimental results of scenario shown in Fig. 10a: (a) Makespan; (b) Speedup.

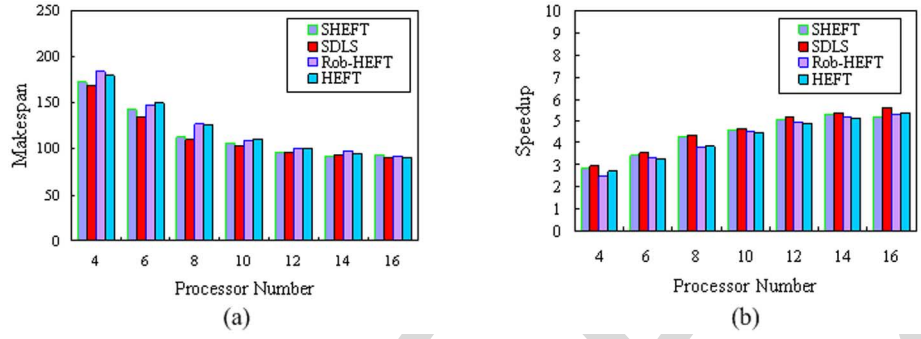

Fig. 12. Experimental results of scenario shown in Fig. 10b: (a) Makespan; (b) Speedup.

low parallelism degree. The other special DAG in Fig. 10b has 50 tasks with 3 levels, which represents a parallel application with very high parallelism degree. Fig. 10b is part of a real-world application DAG graph based on a liquid metal silver solidification processes simulation. This simulation is a classical high performance computing problem and widely used in material and engineering research.

Figs. 11 and 12 show the simulation results of scenario shown in Fig. 10 on heterogeneous cluster systems. From Fig. 11, we observe that the four scheduling algorithms have almost the same performance for low parallelism degree applications. However, from Fig. 12, we observe that the SDLS algorithm has better performance and is suitable for high parallelism degree applications.

8 Conclusions and Future Work

In this paper, we have emphasized the significance of scheduling stochastic parallel applications with precedence constrained tasks on heterogeneous cluster systems. We believe that it is mandatory to design and implement efficient stochastic scheduling algorithms to meet the increasing challenge from real applications with random task processing times and edge communication times on diversified cluster systems with variable computing capabilities. We also mentioned that integration of both expected values and variances of random variables is a key feature which affects the performance of a stochastic scheduling algorithm. We formulated our stochastic task scheduling model and developed effective methods to deal with normally distributed random variables. We proved a lower bound on the expected makespan. We developed a stochastic dynamic level scheduling algorithm SDLS, which employs stochastic bottom level and stochastic dynamic level to produce schedules of high quality.

The performance of the SDLS algorithm was compared with three existing scheduling algorithms, i.e., Rob-HEFT, SHEFT, and HEFT. The comparisons were based on randomly generated application DAGs. The simulation results demonstrate that deterministic scheduling algorithms such as HEFT and DLS are not suitable for stochastic task scheduling. Furthermore, the stochastic scheduling algorithm SDLS is better than Rob-HEFT and SHEFT and has better performance in terms of makespan, speedup, and makespan standard deviation on cluster systems.

Future studies in this area can be conducted in several directions. It will be interesting to extend our stochastic task scheduling algorithm to consider more factors, such as heterogeneous communication links. One can also investigate other variations of the problem, such as parallel tasks, scheduling with deadlines, preemptive scheduling, etc. It will also be interesting to conduct a general investigation to accommodate all probability distributions for stochastic scheduling problem.

Acknowledgments

The authors would like to thank the three anonymous reviewers for their suggestions on improving the paper. This research was partially funded by the Key Program of National Natural Science Foundation of China (Grant 61133005), National Science Foundation of China (Grant Nos. 61070057, 61370098, 61370095), PhD Programs Foundation of Ministry of Education of China (20100161110019), and a project

supported by Scientific Research Fund of Hunan Provincial Education Department (Grant 12A062).

References

[1] M.A. Khan, "Scheduling for Heterogeneous Systems Using Constrained Critical Paths," *Parallel Computing*, vol. 38, no. 4-5, pp. 175-193, Apr./May 2012.

[2] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175-186, Feb. 1993.

[3] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951-967, Sept. 1994.

[4] D. Bozda , F. Özgüner, and U. Catalyurek, "Compaction of Schedules and a Two-Stage Approach for Duplication-Based DAG Scheduling," *IEEE Trans. Parallel and Distributed Systems*, vol. 20, no. 6, pp. 857-871, June 2009.

[5] M.R. Gary and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[6] X. Tang, K. Li, G. Liao, and R. Li, "List Scheduling with Duplication for Heterogeneous Computing Systems," *J. Parallel and Distributed Computing*, vol. 70, no. 4, pp. 323-329, Apr. 2010.

[7] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs onto Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.

[8] Z. Liu, T. Qin, W. Qu, and W. Liu, "DAG Cluster Scheduling Algorithm for Grid Computing," *Proc. IEEE 14th Int'l Conf. Computational Science and Eng. (CSE)*, pp. 632-636, 2011.

[9] K.Y.-K. Kwok and I. Ahmed, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406-471, Dec. 1999.

[10] R.H. Möring, A.S. Schulz, and M. Uetz, "Approximation in Stochastic Scheduling: The Power of LP-Based Priority Policies," *J. ACM*, vol. 46, no. 6, pp. 924-942, Nov. 1999.

[11] N. Megow, M. Uetz, and T. Vredeveld, "Models and Algorithms for Stochastic Online Scheduling," *Math. Operations Research*, vol. 31, no. 3, pp. 513-525, Aug. 2006.

[12] M. Scharbrodt, T. Schickingera, and A. Steger, "A New Average Case Analysis for Completion Time Scheduling," *J. ACM*, vol. 53, no. 1, pp. 121-146, Jan. 2006.

[13] F. Ahmadizar, M. Ghazanfari, and S. Fatemi Ghomi, "Group Shops Scheduling with Makespan Criterion Subject to Random Release Dates and Processing Times," *Computers & Operations Research*, vol. 37, no. 1, pp. 152-162, Jan. 2010.

[14] S. Tongsima, E.H.-M. Sha, C. Chantrapornchai, D.R. Surma, and N.L. Passos, "Probabilistic Loop Scheduling for Applications with Uncertain Execution Time," *IEEE Trans. Computers*, vol. 49, no. 1, pp. 65-80, Jan. 2000.

[15] M. Qiu and E.H.-M. Sha, "Cost Minimization While Satisfying Hard/Soft Timing Constraints for Heterogeneous Embedded Systems," *ACM Trans. Design Automation of Electronic Systems*, vol. 14, no. 2, Article 25, pp. 1-30, Mar. 2009.

[16] M.H. Rothkopf, "Scheduling with Random Service Times," *Management Science*, vol. 12, no. 9, pp. 703-713, May 1966.

[17] G Weiss, "Turnpike Optimality of Smiths Rule in Parallel Machines Stochastic Scheduling," *Math. Operation Research*, vol. 17, no. 2, pp. 255-270, May 1992.

[18] M. Skutella and M. Uetz, "Stochastic Machine Scheduling with Precedence Constraints," *SIAM J. Computing*, vol. 34, no. 4, pp. 788-802, 2005.

[19] J.N. Hagstrom, "Computational Complexity of PERT Problems," *Networks*, vol. 18, no. 2, pp. 139-147, 1988.

[20] L.C. Canon and E. Jeannot, "Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules," Research Report 6895 INRIA, Apr. 2009.

[21] L.C. Canon and E. Jeannot, "Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 4, pp. 532-546, Apr. 2010.

[22] I. Ahmad, Y.-K. Kwok, and M.-Y. Wu, "Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors," *Proc. Int'l Symp. Parallel Architectures, Algorithms, and Networks*, pp. 207-213, June 1996.

[23] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, Mar. 2002.

[24] X. Tang, K. Li, G. Liao, K. Fang, and F. Wu, "A Stochastic Scheduling Algorithm for Precedence Constrained Tasks on Grid," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1083-1091, Aug. 2011.

[25] F. Dong, J. Luo, A. Song, and J. Jin, "Resource Load Based Stochastic DAGs Scheduling Mechanism for Grid Environment," *Proc. 12th IEEE Int'l Conf. High Performance Computing and Comm. (HPCC)*, pp. 197-204, 2010.

[26] J. Gu, X. Gu, and M. Gu, "A Novel Parallel Quantum Genetic Algorithm for Stochastic Job Shop Scheduling," *J. Math. Analysis and Applications*, vol. 355, no. 1, pp. 63-81, Jan. 2009.

[27] E. Ando, T. Nakata, and M. Yamashita, "Approximating the Longest Path Length of a Stochastic DAG by a Normal Distribution in Linear Time," *J. Discrete Algorithms*, vol. 7, no. 4, pp. 420-438, Dec. 2009.

[28] S.C. Sarin, B. Nagarajan, and L. Liao, *Stochastic Scheduling: Expectation-Variance Analysis of a Schedule*. Cambridge Univ. Press, 2010.

[29] O. Sinnen, L. Sousa, and F. Sandnes, "Toward a Realistic Task Scheduling Model," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, no. 3, pp. 263-275, Mar. 2006.

[30] G.H. Hardy, J.E. Littlewood, and G. Polya, *Inequalities*, 2nd ed. Cambridge Univ. Press, 1952.

[31] D. Letić and V. Jevtić, "The Distribution of Time for Clark's Flow and Risk Assessment for the Activities of Pert Network Structure," *Yugoslav J. Operations Research*, vol. 19, no. 1, pp. 195-207, Jan. 2009.

[32] H. Zhao and R. Sakellariou, "An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm," *Proc. 9th Int'l Euro-Par Conf.*, vol. 2790, pp. 189-194, 2003.

[33] A. Colagrossi and M. Landrini, "Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics," *J. Computational Physics*, vol. 191, no. 2, pp. 448-475, Nov. 2003.

[34] C. Xu, L.Y. Wang, and N. Fong, "Stochastic Prediction of Execution Time for Dynamic Bulk Synchronous Computations," *The J. Supercomputing*, vol. 21, no. 1, pp. 91-103, Jan. 2002.

[35] N. Fong, C. Xu, and L.Y. Wang, "Optimal Periodic Remapping of Dynamic Bulk Synchronous Computations," *J. Parallel and Distributed Computing*, vol. 63, no. 11, pp. 1036-1049, Nov. 2003.

[36] C. Clark, "The Greatest of a Finite Set of Random Variables," *Operations Research*, vol. 9, no. 2, pp. 145–162, Mar. 1961.

[37] M.I. Daoud and N. Kharma, "A High Performance Algorithm for Static Task Scheduling in Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing*, vol. 68, no. 4, pp. 399-409, Apr. 2008.

Kenli Li received the PhD degree in computer science from Huazhong University of Science and Technology, China, in 2003, and the MS degree in mathematics from Central South University, Hunan, China, in 2000. He has been a visiting scholar at the University of Illinois at Champaign-Urbana and Urbana from 2004 to 2005. He is now a professor of computer science and technology at Hunan University, China. He is a senior member of CCF. His major research contains parallel computing, grid and cloud computing, and DNA computer.

Xiaoyong Tang received the MS and PhD degrees from Hunan University, China, in 2007 and 2013, respectively. His research interests include modeling and scheduling in distributed computing systems, parallel computing, distributed system reliability, and parallel algorithms. He is a reviewer of TOC, TPDS, JPDC, FGCS, JS, and so on.

Bharadwaj Veeravalli received the BSc degree in physics from Madurai-Kamaraj Uiversity, India, in 1987, the master's degree in electrical communication engineering from Indian Institute of Science, Bangalore, India, in 1991, and the PhD degree from Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India, in 1994. He did his post-doctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering at The National University of Singapore as a tenured associate professor. His mainstream research interests include multiprocessor systems, cluster/grid computing, scheduling in parallel and distributed systems, bioinformatics & computational biology, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He is currently serving on the Editorial Boards of *IEEE Transactions on Computers, IEEE Transactions on SMC-A*, and *International Journal of Computers & Applications*, USA, as an associate editor. He is a Senior Member of IEEE-CS.

Keqin Li is a SUNY distinguished professor of computer science and an Intellectual Ventures endowed visiting chair professor at Tsinghua University, Beijing, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has nearly 290 research publications and has received several Best Paper Awards for his highest quality work. He is currently or has served on the editorial board of *IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers*, *Journal of Parallel and Distributed Computing*, *International Journal of Parallel, Emergent and Distributed Systems*, *International Journal of High Performance Computing and Networking*, and *Optimization Letters*.