Future Generation Computer Systems 27 (2011) 1083–1091

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A stochastic scheduling algorithm for precedence constrained tasks on Grid

Xiaoyong Tang [a,b,*], Kenli Li [b], Guiping Liao [a], Kui Fang [a], Fan Wu [b]

[a] Information Science and Technology College, Hunan Agricultural University, Changsha, China
[b] School of Information Science and Engineering, Hunan University, Changsha, China

ARTICLE INFO

Article history:
Received 30 September 2010
Received in revised form
28 March 2011
Accepted 11 April 2011
Available online 19 April 2011

Keywords:
Stochastic scheduling
Grid
Precedence constrained tasks
Makespan

ABSTRACT

This paper addresses the problems in scheduling a precedence constrained tasks of parallel application with random tasks processing time and edges communication time on Grid computing systems so as to minimize the makespan in stochastic environment. This is a difficult problem and few efforts have been reported on its solution in the literature. The problem is first formulated in a form of stochastic scheduling model on Grid systems. Then, a stochastic heterogeneous earliest finish time (SHEFT) scheduling algorithm is developed that incorporates the expected value and variance of stochastic processing time into scheduling. Our rigorous performance evaluation study, based on randomly generated stochastic parallel application DAG graphs, shows that our proposed SHEFT scheduling algorithm performs much better than the existing scheduling algorithms in terms of makespan, speedup, and makespan standard deviation.

1. Introduction

The popularity of the Internet and the availability of powerful computers and high-speed networks as low-cost commodity components make it possible to construct large-scale high-performance Grid computing systems. These technical opportunities enable the sharing, selection, and aggregation of geographically distributed heterogeneous resources for solving large-scale problems in science, engineering, and commerce [1,2]. To achieve the promising potentials of tremendous distributed resources, effective and efficient scheduling algorithms are fundamentally important. The scheduling problem deals with the coordination and allocation of resources so as to efficiently execute the users' applications. Stochastic Grid parallel applications are submitted by users and generally independent of each other, which request systems' services for their execution. The single parallel application is usually consisted of precedence constrained tasks that they generally require the use of different kinds of resources, e.g., computation, communication (network), storage resources, or specific instruments. A popular representation of a parallel application is the directed acyclic graph (DAG) in which the nodes represent application tasks and the directed arcs or edges represent inter-task dependencies, such as task's precedence. In some cases, weights can be added to nodes and edges to express computational costs and communicating costs, respectively. Scheduling aims at meeting user demands (e.g., in terms of makespan, sum of weighted completion times, maximum lateness, and number of tardy jobs) and the objectives represented by the resource providers (e.g., in terms of profit and resource utilization efficiency), while maintaining a good overall performance (throughput) for the Grid computing systems [3]. It is widely known that the problem of finding the optimal schedule is NP-complete [4] in the general case. Therefore, heuristics can be used to obtain a sub-optimal scheduling rather than parsing all possible schedules. These methods may obtain suboptimal results, but they are much computational cheaper.

List scheduling is a very popular method for precedence constrained task scheduling based on the DAG model. The basic idea of list scheduling is to assign priorities to the tasks of the DAG and place the tasks in a list arranged in descending order of priorities. A task with a higher priority is scheduled before a task with lower priority, and ties are broken using some method. An important issue in DAG scheduling is how to rank (or weigh) the tasks and edges (when communication delay is considered). The rank of a task is used as its priority in the scheduling. Once the tasks and edges are ranked, task-to-resource assignment can be found by considering the following two problems to minimize the makespan: how to parallelize those tasks having no precedence orders in the graph and how to make the time cost along with the critical path in the DAG as small as possible.

There are several effective Grid heuristics scheduling algorithms such as mapping heuristic (MH) [5], dynamic critical path(DCP) [6], levelized-min time (LMT) algorithm [7], dynamic-level scheduling (DLS) algorithms [8], critical-path-on-a-machine

* Corresponding author at: Information Science and Technology College, Hunan Agricultural University, Changsha, China.
E-mail addresses: tang_313@163.com (X. Tang), lkl510@263.net (K. Li), lgpxf@hunau.net (G. Liao), fk@hunau.net (K. Fang), wufan_whu@163.com (F. Wu).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.04.007

(CPOP) algorithms and heterogeneous earliest-finish-time (HEFT) algorithm [9–12]. The HEFT algorithm selects the task with the highest upward rank (an upward rank is defined as the maximum distance from the current node to the exiting node, including the computational cost and communication cost) at each step. The selected task is then assigned to the processor which minimizes its earliest finish time with an insertion-based approach which considers the possible insertion of a task in an earliest idle time slot between two already-scheduled tasks on the same resource. The time complexity of HEFT is $O(m \times v^2)$, where m is the number of processors and v is the number of tasks. HEFT algorithm significantly outperforms DLS, DCP, LMT, MH, and CPOP algorithms in terms of average makespan, speedup, *etc.*, [9].

Most of the above researches assume that parameters such as task processing time and communication time between precedence constrained tasks are fixed and deterministic which are known advance. However, in real-world problems, it usually does not suffice to find good schedules for fixed deterministic processing time, since tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs [13–16]. A natural step to tackle this problem is to consider stochastic scheduling, that is, to interpret processing time and communication time as random variables and to measure the performance of an algorithm by its expected objective value. In this paper, let $V\{v_1, \dots, v_n\}$ be a set of tasks of a parallel application that have to be scheduled on heterogeneous Grid systems with m heterogeneous machines (or unrelated parallel machines) so as to minimize the schedule length (makespan). Any machine can process at most one job at a time, and every job has to be processed on one of the m machines. In contrast to the deterministic version, the crucial assumption is that the task processing time $w(v_i)$ and communication time $w(e_{i,j})$ are not known in advance. Instead, one assumes that the task processing time and communication time are random variables from which we are just given their distribution function. Throughout the paper, task durations are supposed to be stochastically independent.

Since there are a number of scheduling models with different conditions considered in stochastic scheduling [17–19,13,20,14, 21–24], it is convenient to refer to them in the notation of Graham et al. [23] and Allahverdi et al. [24]. Each problem is denoted by the standard three-field notation $\alpha|\beta|\gamma$ with the following intended meaning.

- The field α specifies the machine environment. For instance, $\alpha = P$ denotes the model with identical parallel machines, $\alpha = Q$ denotes the problem where machines have different speeds s_k, and the processing time of task v_i on machine k is $w(v_i)/s_k$, and $\alpha = 1$ is used for problems with a single machine.
- The field β contains the task characteristics. It can be empty, which implies the default of non-preemptive, precedence constrained tasks. Possible entries are, among many others, *prec* for precedence constrained tasks, or *pmtn* for preemptive tasks.
- The field γ denotes the objective function. It is generally a function of the completion times of the tasks. For the total weighted completion time, we write $\gamma = \Sigma w_i C_i$. For the makespan $\gamma = C_{\max}$.

As an example, $Q|v_i \sim \textit{stoch}, \textit{prec}|E[C_{\max}]$ is the stochastic scheduling problem to minimize the makespan of precedence constrained tasks on Grid heterogeneous parallel machines.

Precedence constraints between tasks play a particularly important role in most real-world parallel applications. Therefore, it would be both of theoretical and of practical interest to incorporate those constraints into stochastic scheduling. Considering stochastic tasks with precedence constraints, such as the stochastic DAG model, Skutella and Uetz obtained a performance guarantee of $(1+\varepsilon)\left(1+\frac{m-1}{m\varepsilon}+\max\left\{1, \frac{m-1}{m}\Delta\right\}\right)$ for $P|v_i \sim \textit{stoch}, \textit{prec}|E[w_iC_i]$ (here, ε is an arbitrary constant) [21], where they assume that the squared coefficients of variation of all processing time $w(v)$ are bounded by some constant Δ that is expressed as $\text{Var}(w(v))/E[w(v)]^2 \le \Delta$, for all $v \in V$. For $\Delta \le 1$, the performance guarantees can be simplified to $3+2\sqrt{2}-(1+\sqrt{2})/m$. In most case, the schedule length (makespan) is the key objective of stochastic scheduling. If all tasks are identically and exponentially distributed, Chandy and Reynolds show that a *highest level first* policy minimizes the expected makespan $E[C_{\max}]$ for the special case of two machines and in-tree precedence constraints [22]. Here, the level of a task simply denotes the number of successors in the precedence graph. They also give counterexamples for the case of more than two machines. Bruno extends this result by proving that highest level first even minimizes the makespan stochastically [25]. Pinedo and Weiss extend Chandy and Reynolds result to the case where the exponential processing time distributions of tasks of different levels may be different [26]. For the case of more than two machines, Papadimitriou and Tsitsiklis prove asymptotic optimality of a highest level first policy [27]. Their assumptions are that all processing times are independent and identically distributed, and this distribution has finite moments of any order. This is particularly true for the exponential distribution. Although somewhat counter-intuitive, results for out-trees (out-forests) seem to be much more difficult to obtain: Coffman and Liu show that a preemptive highest level first policy is optimal to minimize the expected makespan for some rather restricted scenarios [28]. These results, however, are restricted to somewhat artificial precedence constrained tasks (such as in-tree precedence constraints) and execution on identical machines.

To the best of our knowledge, no results of scheduling algorithm were previously known for $Q|v_i \sim \textit{stoch}, \textit{prec}|E[C_{\max}]$ on Grid with m heterogeneous machines. In recognition of this, we drive a stochastic scheduling model for this problem. In order to effectively scheduling precedence constrained stochastic tasks, we propose a stochastic heterogeneous earliest finish time (SHEFT) scheduling algorithm, which incorporate the stochastic attribute, such as expected value and variance, of task processing time and edge communication time into scheduling. At last, we establish a stochastic scheduling simulation experiment platform and compare SHEFT with two well-known scheduling algorithms HEFT and DCP.

The rest of the paper is organized as follows: In Section 2, we describe the definitions and background of stochastic scheduling problem on Grid computing systems. In Section 3, we propose a SHEFT scheduling algorithm. In Section 4, we verify the performance of the proposed algorithm by comparing the results obtained from performance evaluation and conclude the paper in Section 5.

2. Definitions and background

2.1. The Grid scheduling architecture

A Grid is a system of high diversity, which geographically distributed sites interconnect through WAN. We define a site as a location that contains many computing resources of different processing capabilities. Heterogeneity and dynamicity cause resources in grids to be distributed or in clusters rather than uniformly. We can also generalize a scheduling process in the Grid into two stages: resource discovering and filtering and resource selecting and scheduling according to certain objectives. As a study of scheduling algorithms is our primary concern here, we focus on the second step. Based on these observations, Fig. 1 depicts a model of Grid scheduling systems.

As Grid resources are under the control of local schedulers, which provide access point to the resources, the central Grid scheduler must be built on top of the existing local schedulers. Basically, a central Grid scheduler (GS) receives applications

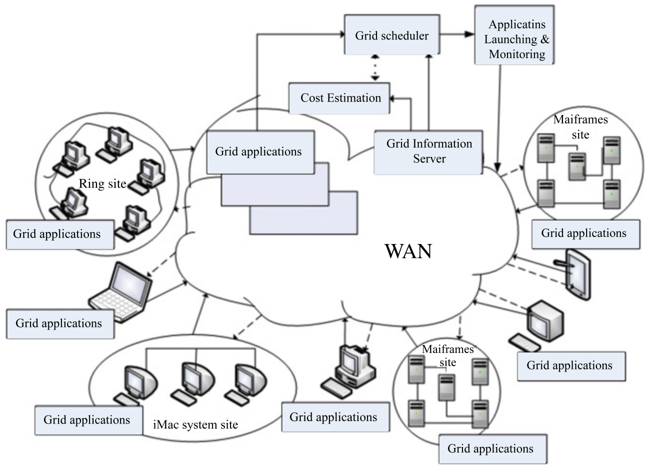

Fig. 1. Grid scheduling architecture.

from Grid users, selects feasible resources for these applications according to acquired information from the Grid Information Service module, and finally generates application-to-resource mappings, based on certain objective functions and predicted resource performance [29]. The central Grid scheduler is simpler to implement, easier to manage, and quicker to repair in case of a failure. Information about the status of available resources is very important for a Grid scheduler to make a proper schedule, especially when the heterogeneous and dynamic nature of the Grid is taken into account. The role of the Grid information service (GIS) is to provide such information to Grid schedulers. GIS is responsible for collecting and predicting the resource state information, such as CPU capacities, memory size, communication capacity, software availabilities, and load of a site in a particular period [29]. The launching and monitoring (LM) module implements a finally-determined schedule by submitting applications to selected resources, staging input data and executables if necessary, and monitoring the execution of the applications [29].

In our scheduling model, the Grid system is modeled as an undirected graph $GT = \langle R, L\rangle$, R is a finite set of p vertices, and L is a finite set of virtual fully connected undirected edges or arcs [11]. A vertex p_k represents the machine k with different processing capabilities, such as computation capacity s_k (measured in millions instruction per second, MIPS), and also includes storages, programs, *etc.* An undirected edge $l_{n,k}$ represents a bidirectional communication link between the incident machines p_n and p_k, which represents not only a physical connection, but also a logical link. The advantage of virtualization is to simplify the graph model for Grid computing systems. The weight $w(l_{n,k})$ assigned to a link $l_{n,k}$ stands for its communication capacity (the amount of data that can go through the link in a unit time). We further assume that all inter-machine communications are performed without contention, and the communication overhead between two tasks scheduled on the same machine is taken as zero. This assumption holds since our computing environment consists of machines connected with wide area network links as pointed out in the literature [11,30].

2.2. Stochastic parallel application model

Task graphs are encountered in many models used in operations research and resource management, such as scheduling, computer, and network models. Graph models are often used also to study activities that contain some concurrency. For example, a DAG can describe parallel applications where nodes represent tasks and directed arcs represent synchronization constraints and communication among tasks. Generally, a stochastic parallel application is

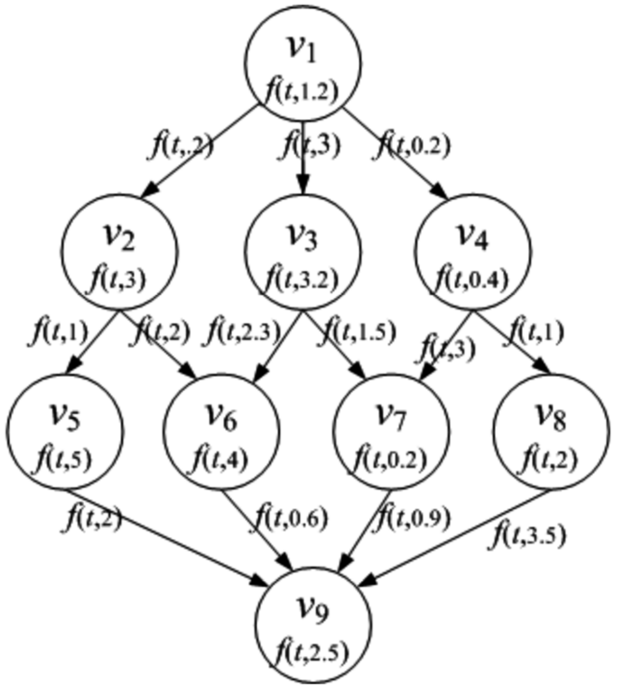

Fig. 2. An example of stochastic parallel application with exponential distribution.

represented by a DAG $G = \langle V, E\rangle$, where V is the set of v tasks that can be executed on any of the available machines; $E \in V \times V$ is the 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. A task may have one or more inputs. When all its inputs are available, the task is triggered to execute. After its execution, it generates its outputs. 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. In our stochastic setting, the processing time and communication time are random, assumed to be independent and exponentially distribution or normal distribution [20,14,26,27]. For each application submission, the user needs to estimate these probabilities or distributions, which can be obtained by building a historic table and using statistical profiling [14,16]. We think that statistical profiling can be used to determine probabilities in many applications in the Grid area.

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 node $v \in V$ without predecessors, $pred(v_i) = \phi$, is called an *entry task* and if it is without successors, $succ(v_i) = \phi$, it is named *exit task*. Without losing of generality, we assumed that the DAG has exactly one entry task v_{entry} and one exit task v_{exit}. If multiple exit tasks or entry tasks exist, they may be connected with zero time-weight edges to a single pseudo-exit task or a single entry task that has zero time-weight. Fig. 2 shows an example DAG with assigned task and edge which is exponential distribution.

2.3. Scheduling attributes

The main difference between stochastic scheduling and deterministic scheduling is that processing time and communication time are deterministic or random. Thus, how to compute the probability distribution of tasks in the DAG graph is the key technique to stochastic scheduling. In this paper, the processing time and communication time are assumed to follow exponentially distributions, which is reasonable to most real-world problems [14,26,27].

In probability theory, the expected value of a random variable indicates its average or central value and reflects its most important attribute. However, the real value of random variable does not equal to its expected value in most case. The other important attribute of a random variable is variance, which is the expected value of the square of the deviation of that variable from its expected value. Thus, the variance is an important measure of the amount of variation within the values of that variable, taking account of all possible values and their probabilities. The variance is first examined by Möring and Schulz in stochastic scheduling problem and got a good performance approximation in paper [23]. At the same time, Skutella and Uetz pointed out that the real processing time of stochastic task is affected by the standard deviation of random variable. They prove that the system scheduling performance are bounded by some constant Δ that is expressed as $\mathrm{Var}(X)/E[X]^2 \leq \Delta$ [26]. For $\Delta \leq 1$, the scheduling performance are mostly affected by sum of the random variable's expected value and square of variance. For others, the scheduling performance increases as random variable's variance decreases. Thus, we take the variance of random variable into account and the approximate weight of random variable $Aw(X)$ is defined as

$$Aw(X) = \begin{cases} E[X] + \sqrt{\mathrm{Var}(X)} & \mathrm{Var}(X)/E[X]^2 \leq 1 \\ E[X]\left(1 + \dfrac{1}{\sqrt{\mathrm{Var}(X)}}\right) & \text{Others.} \end{cases} \tag{1}$$

As the processing time of tasks and communication time of edges in stochastic DAG are assumed to follow exponentially distributions, each random variable are satisfied with $\mathrm{Var}(X)/E[X]^2 = 1$. Thus, the approximate weight of random variable with exponential distribution is $Aw(X) = E[X] + \sqrt{\mathrm{Var}(X)}$. For example, the approximate weight of task v_2 in Fig. 1 is $Aw(v_2) = 0.67$.

Before presenting the objective function, it is necessary to define the approximate earliest execution start time $AS(v_i, p_k)$ and the approximate earliest execution finish time $AF(v_i, p_k)$ of task v_i on machine p_k. For the entry task v_{entry}

$$AS(v_{entry}, p_k) = 0. \tag{2}$$

For the other tasks in the graph, the approximate earliest start time and earliest finish time are computed recursively, starting from the entry task, as shown in (3) and (4), respectively. In order to compute the approximate earliest start time for a task v_i, all immediate predecessor tasks of v_i must have been scheduled:

$$AS(v_i, p_k) = \max\{Available(p_k), \max_{v_j \in pred(v_i)}\{AF(v_j, p_n) + Aw(e_{j,i})\}\} \tag{3}$$

$$AF(v_i, p_k) = AS(v_i, p_k) + \frac{Aw(v_i)}{s_k} \tag{4}$$

where $Aw(e_{j,i})$ is the approximate weight of edge $e_{j,i}$'s communication time which transferring data from task v_j (scheduled on p_n) to task v_i(scheduled on p_k) and is computed by Eq. (1). When both v_j and v_i are scheduled on the same machine, $Aw(e_{j,i})$ becomes zero, since we assume that the intramachine communication cost is negligible when it is compared with the intermachine communication cost. The $pred(v_i)$ is the set of immediate predecessor tasks to task v_i, and $Available(p_k)$ is the expected earliest time at which machine p_k is ready for task execution. After all tasks in a graph are scheduled, the schedule length (i.e., overall completion time) will be the actual finish time of the exit task v_{exit}; thus, the schedule length (which is also called makespan) is defined as follows:

$$makespan = EFF(v_{exit}) \tag{5}$$

Table 1
Definitions of notations.

Notation	Definition
V	A set of v stochastic tasks in the application
v_i	The ith stochastic task in the application
$w(v_i)$	The probability distribution of task v_i processing time
E	A set of directed edges representing communication among tasks in V
$e_{i,j}$	The directed edge from ith task to jth task
$w(e_{i,j})$	The probability distribution of edge $e_{i,j}$ communication time
P	A set of m machines
p_k	The kth machine in Grid
s_k	The kth machine computation capacity
$w(l_{n,k})$	The direct communication capacity between machines p_n and p_k
$succ(v_i)$	A set of immediate successors of task v_i
$pred(v_i)$	A set of immediate predecessors of task v_i
$AS(v_i, p_k)$	The expected earliest execution start time of task v_i on machine p_k
$AF(v_i, p_k)$	The expected earliest execution finish time of task v_i on machine p_k
$Aw(X)$	The approximate weight of random variable X

where the $EFF(v_{exit})$ is the earliest actual execution finish time of task v_{exit} and different from the approximate earliest finish time $AF(v_{exit})$. The objective function of the task-scheduling problem is to determine the assignment of tasks of a given application to machines such that its scheduling length is minimized. For ease of understanding, we summarize the notations and their meanings used throughout this paper in Table 1.

3. The proposed stochastic list scheduling

This section presents an algorithm for list scheduling on Grid called SHEFT, which aims to achieve high performance and low complexity. The algorithm consists of two mechanisms, a listing mechanism, which is a modified version of the HEFT [9–11] heuristic scheduling algorithm and a machine assignment mechanism which each task (in order of its priority) is assigned to a machine that minimizes the makespan. The pseudo code of the SHEFT algorithm is shown in Algorithm 1. First, we outline the concept of stochastic task priority. Then, we propose and analyze the stochastic list scheduling algorithm.

```
1  Compute Srank value for all tasks use Eq. (6) by traversing
   application graph, starting from the exit task
2  Sort the tasks in a scheduling list by non-increasing order of
   Srank value
3  while there are unscheduled tasks in the list do
4      Remove the first task $v_i$ from the scheduling list
5      for each machine $p_k$ in the machine set ($p_k \in P$) do
6          Compute the approximate earliest start time use
           Eq. (3)
7          Compute the approximate earliest finish time by
           Eq. (4)
8      end
9      find the minimum approximate earliest finish time
       machine $p_n$
10     Assign task $v_i$ to machine $p_n$ with minimize $AF(v_i, p_n)$
11 end
```

Algorithm 1: The pseudo code for SHEFT algorithm.

3.1. Stochastic task priorities phase

Our stochastic list scheduling algorithm will use stochastic upward rank (*Srank*) attribution to compute tasks priorities. The *Srank* is explained in Definition 1.

Definition 1. Given a stochastic parallel application DAG with v tasks and e edges and Grid computing system with m

heterogeneous machines, the *Srank* during a particular scheduling step is a rank of task, from an exit task to itself, which has the sum of approximate task processing time and approximate edge communication time over all machines. Approximate communication time between tasks scheduled on the same machine are assumed to be zero.

The *Srank* is recursively defined as follows:

$$Srank(v_i) = \frac{\sum_{k=1}^{m} \frac{Aw(v_i)}{s_k}}{m} + \max_{v_j \in \mathrm{succ}(v_i)} \left\{ \frac{Aw(e_{i,j})}{\overline{w(l)}} + Srank(v_j) \right\} \quad (6)$$

where $succ(v_i)$ is the set of immediate successors of task v_i, $\frac{\sum_{k=1}^{m} \frac{Aw(v_i)}{s_k}}{m}$ is the approximate mean stochastic rank of task v_i in Grid which take the expected value and variance of task v_i into account, $\overline{w(l)}$ is the mean communication capacity in the Grid system. The rank is computed recursively by traversing the task graph upward, starting from the exit task. For the exit task v_{exit}, the *Srank* value is equal to

$$Srank(v_{exit}) = \frac{\sum_{k=1}^{m} \frac{Aw(v_{exit})}{s_k}}{m}. \quad (7)$$

Basically, $Srank(v_i)$ is the length of the stochastic critical path from task v_i to the exit task, including the approximate mean stochastic rank of task v_i.

3.2. Machine selection phase

In machine selection phase, unscheduled task in the task sequence is selected and scheduled on a machine that can complete its execution with minimize approximate earliest finish time $AF(v_i, p_n)$. To achieve this goal, SHEFT arranges task scheduling sequence by *Srank* that takes the attributes of random variable into account. For each stochastic task, SHEFT computes $AF(v_i, p_n)$ by using Eqs. (3) and (4). These ideas are implemented from step 5 to 8. In step 9, SHEFT finds a machine with minimize approximate earliest finish time $AF(v_i, p_n)$ and assigns task v_i to this machine p_n in step 10. It is to be noted that this machine may or may not be its best-suited machine due to the stochastic of task. This type of heuristic automatically takes into consideration the computing power of the candidate machine.

3.3. Algorithm complexity analysis

The time complexity of scheduling algorithms for parallel application DAG is usually expressed in terms of number of task v, number of edges e, and number of machines m. The time-complexity of SHEFT is analyzed as follows. Computing the *Srank* of task can be done in time $\max\{O(ve), O(vm)\}$, and sorting the tasks can be done in time $O(v \log v)$. The machine selection for all tasks can be done in time $O(v|v|m)$, and $|v|$ is the max degree of v_i in application DAG. In Grid systems, the number of machines may be larger than the number of application's edges in most case. Thus, the complexity of the algorithm SHEFT is $O(v|v|m)$.

4. Performance evaluation

In this section, we compare the performance of the SHEFT algorithm with two well-known scheduling algorithms in Grid systems: the HEFT [9] and DCP [6] algorithms. To make the comparison efficient, we turn the stochastic scheduling problem into deterministic one by giving the expected value as the weight of task for HEFT and DCP algorithms. The comparison is intended not only to present quantitative results, but also to qualitatively analyze the results and to suggest explanations, for a better insight in the overall scheduling problem.

To test the performance of these algorithms, we have built an extensive simulation environment of Grid systems with 16 heterogeneous machines that computation capacities varies from Pentium II to Pentium IV. In order to interconnect these machines, a network with switches is employed, where the network topology is randomly generated and each machine is randomly connected to a switch. The communication capacity of links are assumed to be uniformly distributed between 10 and 100 Mbits/s. Stochastic parallel application graphs are based on randomly generator by varying parameters such as *DAG size and height of the DAG* (h). The performance metrics chosen for the comparison are the schedule length (*makespan* Eq. (5)), the speedup, and the makespan standard deviation. The speedup is computed by dividing the sequential execution time (i.e., cumulative execution time) by the parallel execution time (i.e., the makespan of the output schedule) as shown in Eq. (8):

$$Speedup = \frac{\sum_{v_i \in V} ET(v_i)}{makespan} \quad (8)$$

where $ET(v_i)$ is the actual execution time of task v_i. The sequential execution time is computed by assigning all tasks to a single machine that minimizes the cumulative of the computation costs. If the sum of the computational costs is maximized, it results a higher speedup, but ends up with the same rank of the scheduling algorithms.

The other performance metric is makespan standard deviation. Intuitively, the standard deviation of the makespan distribution tells how narrow the makespan distribution is. The narrower the distribution, the smaller the standard deviation is. This metric is examined in this paper because when you are given two schedules the one for which the standard deviation is smaller is the one for which realizations are more likely to have a stable performance in the Grid system.

4.1. Randomly generated application graphs

In our study, we considered the randomly generated stochastic application graphs. A random graph generator was implemented to generates stochastic application DAGs with various characteristics that depend on serval input parameters given below. 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 the stochastic application DAGs, which is followed by the execution of the scheduling algorithms to generate output schedules, and, finally, it computes the performance metrics based on the schedules. For the generation of random graphs, which are commonly used to compare scheduling algorithms [8,9], three fundamental characteristics of DAG are considered:

- *DAG size*, v: The number of tasks in the application DAG.
- *Height of the DAG*(h): The v tasks are randomly partitioned into h levels.
- The maximum and minimum λ value $(\lambda_{\max}, \lambda_{\min})$ of task processing time and communication time between tasks with exponentially distribution: The λ value is a uniform random variable on the interval $(\lambda_{\max}, \lambda_{\min})$.

In our simulation experiments, graphs are generated for all combinations of the above parameters with number of tasks ranged between 50 and 300. Every possible edge (DAGs are acyclic) is created with the same probability, 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 for these graphs.

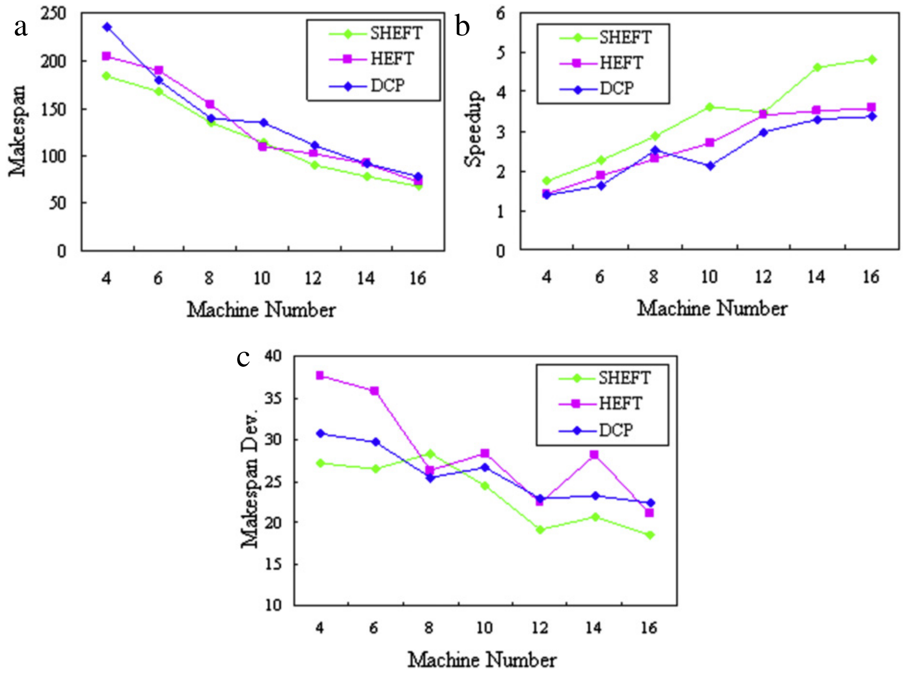

Fig. 3. Performance impact of 100 tasks: (a) makespan in seconds; (b) speedup; (c) makespan standard deviation.

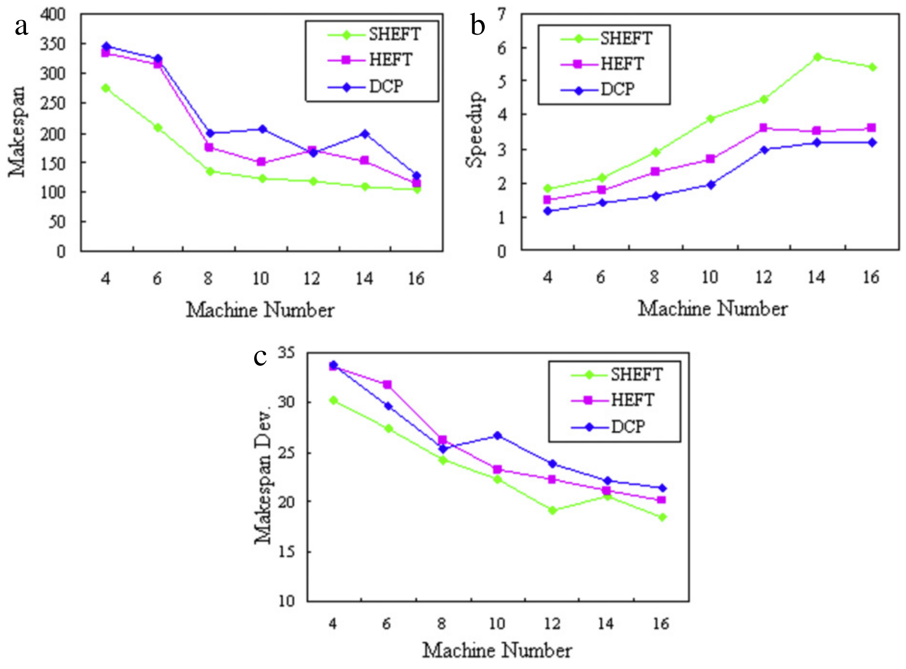

Fig. 4. Performance impact of 200 tasks: (a) makespan in seconds; (b) speedup; (c) makespan standard deviation.

4.2. Experimental results

The goal of the experiments is to compare the proposed SHEFT algorithm with HEFT and DCP. In order to effective simulate the real-world stochastic environment, the actually execution time of task is evaluated by giving their random probability distribution. To make the comparison fair, each schedule of scheduling strategy has the same task random probability.

For the first set of simulation studies, we compare and analyse the sensitivity of machine number with 100, 200, and 300 tasks respectively. The results are shown in Figs. 3–5. For performance metrics of makespan and speedup, each data point is the average of the data obtained in 500 experiments, and for the makespan standard deviation metric, the data are from these 500 experiments. Fig. 3 shows the simulation results of the SHEFT, HEFT, and DCP algorithms on Grid system for stochastic parallel application DAG with 100 tasks. We observe from Fig. 3 that the SHEFT outperforms HEFT and DCP by (10.7%, 16.1%) in terms of the average makespan, (19.7%, 26.1%) in terms of the average speedup, and (21.3%, 9.97%) in terms of makespan standard deviation, respectively. This is mainly due to the fact that the SHEFT algorithm is a stochastic scheduling strategy and considers the stochastic attribute, such as expected value and variance, in scheduling algorithm, which produce a good schedule for stochastic scheduling problem $Q|v_i \sim stoch, prec|E[C_{\max}]$. However, the HEFT and DCP algorithms are deterministic and

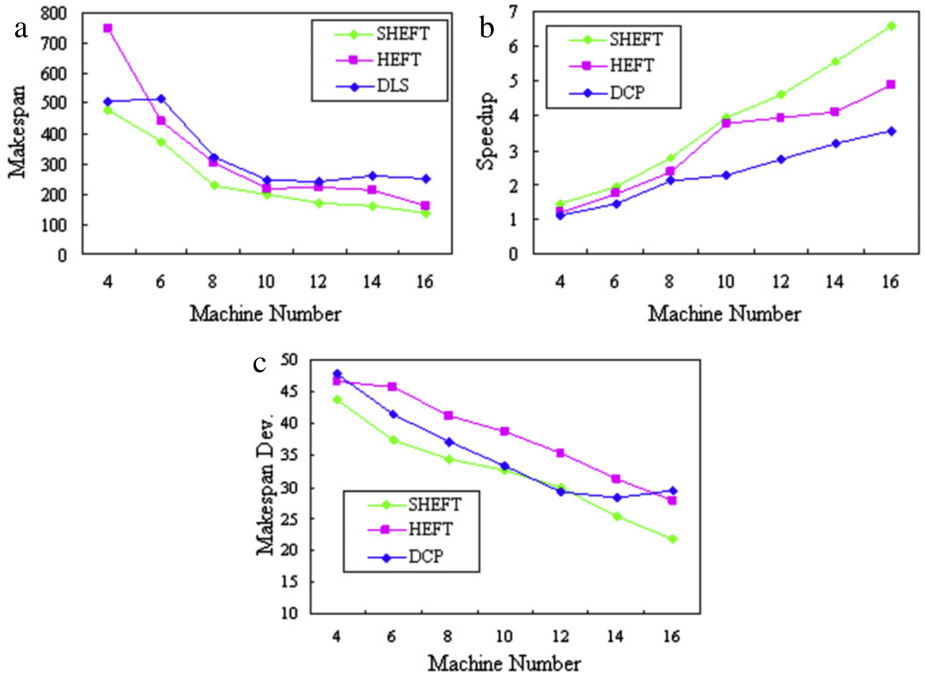

Fig. 5. Performance impact of 300 tasks: (a) makespan in seconds; (b) speedup; (c) makespan standard deviation.

ignore the available stochastic information about task processing time and communication time among tasks, which merely select a machine for a task with the expected value of variables. For example, the task processing time follows exponentially distribution as $f(t) = 0.1e^{-0.1t}$, the expected value is 10, and the variance is 100. In this situation, the scheduling selections of HEFT and DCP are according to the value of 10, which is far from the actual execution time of task. However, the SHEFT schedules the task according to the value of 20, which is close to the actual execution time.

As the comparison between HEFT and DCP of simulation results, we find that HEFT is better than DCP in terms of makespan and speedup. This is mainly due to the fact that HEFT can more effectively compute the critical task than DCP in DAG. However, for makespan standard deviation, DCP is better than HEFT. This phenomena can attribute to the fact that the variance of DCP is better than HEFT. As the machine number increases, the makespan and makespan standard deviation of three algorithms decreases, and the speedup of them increases.

Fig. 4 shows the simulation results for stochastic parallel application DAG with 200 tasks. We observe from Fig. 4 that the SHEFT significantly outperforms HEFT by (30.7%, 39.05%, 9.8%) and DCP by (46.2%, 49.96%, 12.5%) in terms of the average makespan, speedup, and makespan standard deviation, respectively. However, the improvements of SHEFT over HEFT and DCP are almost at the same level except that the machine number is 14, which reflect the intricacy of stochastic scheduling problem. The improvements of SHEFT over HEFT and DCP could also be concluded from Fig. 5. Fig. 5 shows the simulation results for stochastic parallel application DAG with 300 tasks, SHEFT outperforms HEFT by (32%, 21.6%, 18.6%), and DCP by (34.3%, 29.1%, 9.7%) in terms of average makespan, speedup, and makespan standard deviation, respectively.

To examine the performance sensitivity of the three scheduling algorithms SHEFT, HEFT, and DCP to the stochastic parallel application DAG size, in this set of experiments, we vary DAG size from 50 to 300 with step 50. The results reported in Fig. 6 for 10 machines and Fig. 7 for 16 machines reveal that SHEFT outperform HEFT and DCP in terms of makespan, speedup, and makespan standard deviation. As the DAG size increases, the improvement becomes more significant. The above simulation results also show a fact that the deterministic scheduling algorithm (such as HEFT and DCP) is not suitable for stochastic scheduling problem $Q|v_i \sim stoch, prec|E[C_{\max}]$.

5. Conclusions and future work

In this paper, we attempt to incorporate stochastic parallel application into task scheduling on Grid systems. We believe that it is mandatory to design and implement stochastic scheduling to meet the random of task processing time and edge communication time, which is a key attribute to the performance of stochastic scheduling problem $Q|v_i \sim stoch, prec|E[C_{\max}]$. The problem is first formulated in a form of Grid system model and stochastic parallel application DAG. To solve the problem efficiently, a stochastic heterogeneous earliest finish time (SHEFT) scheduling algorithm is proposed that is a modified version of the deterministic scheduling algorithm (HEFT) and incorporate the stochastic attribute of task processing time and edge communication time into scheduling. The performance of stochastic heterogeneous earliest finish time scheduling algorithm was compared with deterministic HEFT and DCP. The comparisons were based on randomly generated application DAGs. The simulation experimental results clearly demonstrate that our proposed algorithm SHEFT outperforms the existing well-known scheduling algorithms HEFT and DCP in terms of minimizing the makespan, makespan standard deviation, and improving the speedup.

This work is one of the first attempts to investigate scheduling precedence constrained stochastic tasks problem into Grid system. Future studies in this domain are threefold: first, it will be interesting to extend our stochastic scheduling algorithm to consider more other constraints, such as deadline, release time *etc.*; second, it will be interesting to study the problem context under large-scale heterogeneous distributed computing systems with security and reliability requirements; third, how the average execution time and standard deviation of tasks are obtained is a challenging and interesting problem, we plan to study it in our future work.

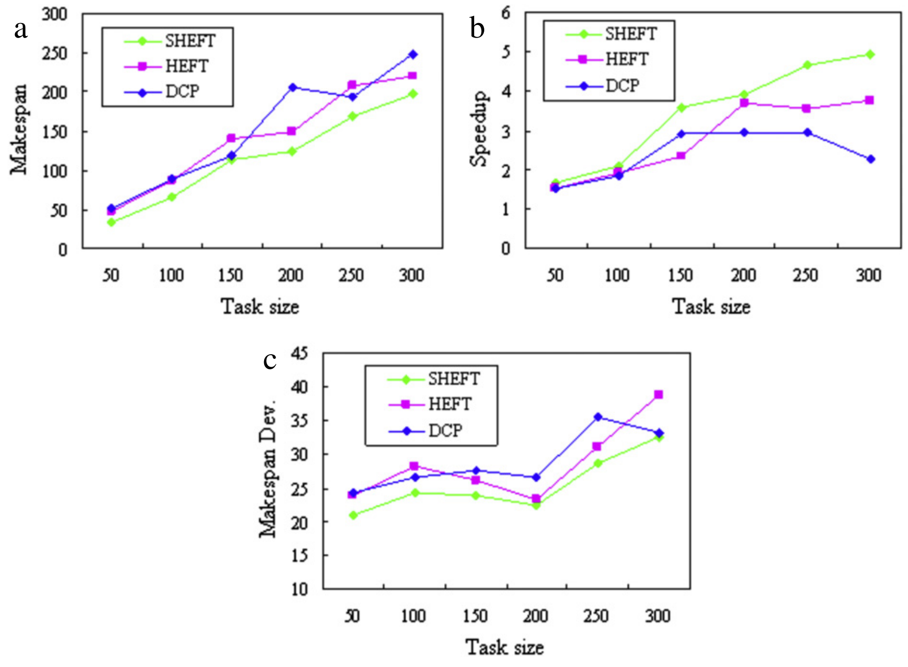

Fig. 6. Performance impact of 10 machines: (a) makespan in seconds; (b) speedup; (c) makespan standard deviation.

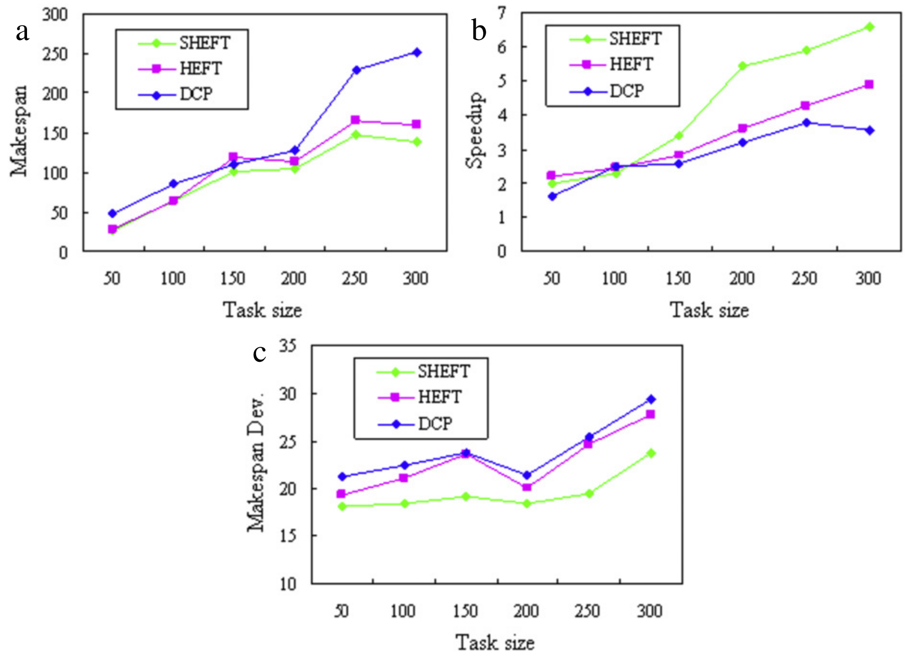

Fig. 7. Performance impact of 16 machines: (a) makespan in seconds; (b) speedup; (c) makespan standard deviation.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61070057), the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (Grant No. 708066), the Hunan provincial Science and Technology important Projects (Grant No. 2010FJ1006), the Hunan Provincial Innovation Foundation For Postgraduate (Grant No. CX2010B139), and A Project Supported by Scientific Research Fund of Hunan Provincial Education Department (08C435).

References

[1] M. Wieczoreka, A. Hoheiselb, R. Prodana, Towards a general model of the multi-criteria workflow scheduling on the grid, Future Gener. Comput. Syst. 25 (3) (2009) 237–256.
[2] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, 1999.
[3] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, E. Varvarigos, A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in grid networks, Comput. Commun. 32 (8) (2009) 1172–1184.
[4] M.R. Gary, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., San Francisco, CA, 1979.
[5] H. El-Rewini, T.G. Lewis, Scheduling parallel program tasks onto arbitrary target machines, J. Parallel Distrib. Comput. 9 (2) (1990) 138–153.

[6] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs onto multiprocessors, IEEE Trans. Parallel Distrib. Syst. 7 (5) (1996) 506–521.
[7] M. Iverson, F. Ozuner, G. Follen, Parallelizing existing applications in a distributed heterogeneous environment, in: Proceedings of Heterogeneous Computing Workshop, 1995, pp. 93–100.
[8] G.C. Sih, E.A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous machine architectures, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 175–187.
[9] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[10] X. Tang, K. Li, D. Padua, Communication contention in APN list scheduling algorithm, Sci. China Ser. F 52 (1) (2009) 59–69.
[11] X. Tang, K. Li, G. Liao, R. Li, List scheduling with duplication for heterogeneous computing systems, J. Parallel Distrib. Comput. 70 (4) (2010) 323–329.
[12] X. Tang, K. Li, R. Li, B. Veeravalli, Reliability-aware scheduling strategy for heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 70 (9) (2010) 941–952.
[13] R.H. Möring, A.S. Schulz, M. Uetz, Approximation in stochastic scheduling: the power of LP-based priority policies, J. ACM 46 (6) (1999) 924–942.
[14] Mark Scharbrodt, Thomas Schickingera, Angelika Steger, A new average case analysis for completion time scheduling, J. ACM 53 (1) (2006) 121–146.
[15] S. Tongsima, E.H.-M. Sha, C. Chantrapornchai, D.R. Surma, N.L. Passos, Probabilistic loop scheduling for applications with uncertain execution time, IEEE Trans. Comput. 49 (1) (2000) 65–80.
[16] M. Qiu, E.H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Trans. Des. Autom. Electron. Syst. 14 (2) (2009) 1–30.
[17] M.H. Rothkopf, Scheduling with random service times, Manage. Sci. 12 (9) (1966) 703–713.
[18] G. Weiss, Approximation results in parallel machines stochastic scheduling, Ann. Oper. Res. 26 (1) (1990) 195–242.
[19] G. Weiss, Turnpike optimality of Smiths rule in parallel machines stochastic scheduling, Math. Oper. Res. 17 (2) (1992) 255–270.
[20] N. Megow, M. Uetz, T. Vredeveld, Models and algorithms for stochastic online scheduling, Math. Oper. Res. 31 (3) (2006) 513–525.
[21] Martin Skutella, Marc Uetz, Stochastic machine scheduling with precedence constraints, SIAM J. Comput. 34 (4) (2005) 788–802.
[22] K.M. Chandy, P.F. Reynolds, Scheduling partially ordered tasks with probabilistic execution times, Oper. Syst. Rev. 9 (1975) 169–177.
[23] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[24] A. Allahverdi, J.N.D. Gupta, T. Aldowaisan, A review of scheduling research involving setup considerations, Manage. Sci. 27 (1999) 219–239.
[25] J. Bruno, On scheduling tasks with exponential service times and in-tree precedence constraints, Acta Inform. 22 (1985) 139–148.
[26] M. Pinedo, G. Weiss, Scheduling jobs with exponentially distributed processing times and in tree precedence constraints on two parallel machines, Oper. Res. 33 (1985) 1381–1388.
[27] C.H. Papadimitriou, J.N. Tsitsiklis, On stochastic scheduling with in-tree precedence constraints, SIAM J. Comput. 16 (1987) 1–6.
[28] E.G. Coffman, Z. Liu, On the optimal stochastic scheduling of out-forests, Oper. Res. 40 (1992) 867–875.
[29] F. Dong, Selim G. Akl, Scheduling algorithms for grid computing: state of the art and open problems, Technical Report No. 2006-504, 2006.
[30] M. Kalantari, M. Akbaria, A parallel solution for scheduling of real time applications on grid environments, Future Gener. Comput. Syst. 25 (7) (2009) 704–716.

Xiaoyong Tang received his master degree from Hunan University, China, in 2007. He is currently working towards the Ph.D. degree at Hunan University of China. His research interests include modeling and scheduling for distributed computing systems, distributed system reliability, distributed system security, and parallel algorithms.

Kenli Li received the Ph.D.degree in computer science from Huazhong University of Science and Technology, China, in 2003, and the B.S. degree in mathematics from Central South University, China, in 2000. He has been a visiting scholar at University of Illinoise at Champaign and Urbana from 2004 to 2005. He is now a professor of Computer science and Technology at Hunan University. He is a senior member of CCF. His major research contains parallel computing, Grid and Cloud computing, and DNA computer.

Guiping Liao received the M.S. degree in ecology from Institute of Applied Ecology Research, Chinese Academy of Sciences, China, in 1994, and Ph.D. degree in agronomy from Hunan Agricultural University, China, in 2000. He was a Post-Doctor of School of Computer, National University of Defense technology, China, from November, 2004 to November, 2005. He was a visiting scientist of the Plant Biotechnology Institute (PBI) of National Research Council, Canada, from November, 2004 to November, 2005. He is currently a director of Institute of Agricultural Information technology, and a professor and Ph.D. supervisor of School of Information Science and Technology, Hunan Agricultural University. His research interests include expert system, intelligent information processing, distributed computing systems, and computer vision.

Kui Fang received the Ph.D. degree in computer science and Technology from National University of Defense Technology, China, in 2000, and the M.S. degree in Computational mathematics from Xi an Jiaotong University, China, in 1985. He is now a professor of Computer science and Technology at Hunan Agricultural University. His major research include intelligent information processing, computer graphics, parallel computing.

Fan wu received his master degree from Wuhan University, China, in 2004. He is currently working towards the Ph.D. degree at Hunan University of China. His research interests include parallel computing and DNA computing.