J Grid Computing (2013) 11:47–61
DOI 10.1007/s10723-012-9237-0

Dependable Grid Workflow Scheduling Based on Resource Availability

Yongcai Tao · Hai Jin · Song Wu · Xuanhua Shi · Lei Shi

Received: 14 September 2011 / Accepted: 13 September 2012 / Published online: 29 September 2012

Abstract Due to the highly dynamic feature, dependable workflow scheduling is critical in the Grid environment. Various scheduling algorithms have been proposed, but seldom consider the resource reliability. Current Grid systems mainly exploit fault tolerance mechanism to guarantee the dependable workflow execution, which, however, wastes system resources. The paper proposes a dependable Grid workflow scheduling system (called DGWS). It introduces a Markov Chain-based resource availability prediction model. Based on the model, a reliability cost driven workflow scheduling algorithm is presented. The performance evaluation results, including the simulation on both parametric randomly generated DAGs and two real scientific workflow applications, demonstrate that compared to present workflow scheduling algorithms, DGWS improves the success ratio of tasks and diminishes the makespan of workflow, so improves the dependability of workflow execution in the dynamic Grid environments.

Keywords Grid · Workflow scheduling · Dependability · Markov

Y. Tao (✉) · L. Shi
School of Information Engineering,
Zhengzhou University, Zhengzhou,
Henan 450000, China
e-mail: ieyctao@zzu.edu.cn

H. Jin · S. Wu · X. Shi
Services Computing Technology and System Lab,
Cluster and Grid Computing Lab,
Huazhong University of Science and Technology,
Wuhan, 430074, China

1 Introduction

Grid workflow is a complex and typical Grid application and opens up a new avenue for complex and collaborative scientific research. The workflow scheduling is an NP-complete problem, and many heuristics have been proposed [1, 2]. However, due to the diverse failures and error conditions in the Grid environments, resource failure is increasingly becoming severe and poses great challenges to the Grid workflow scheduling. For example, most Grid resources are non-dedicated and can enter and depart without any prior notice. In addition, the change of resource local policy, the breakdown of software and hardware and the malfunction of network fabric can result in resource inaccessibility. Hence, jobs fail frequently and QoS can't be guaranteed. Present heuristic algorithms (e.g., HEFT [1], CPOP [1], DLS [3]) rarely consider the resource reliability and current systems generally resort to the fault recovery mechanism [4], such as checkpoint/restart, replication, primary/backup, etc.

Although relieving the challenges to some extent, the mechanism sacrifices system resources. For example, checkpoint/restart policy requires extra disk space and network bandwidth to record the job running information. Replication and primary/backup policies are to run the job at multiple available resources. Moreover, the fault recovery mechanism belongs to compensating methodology and can't prevent job failures in advance. To prevent the job failures proactively, the accurate information of temporal and spatial distribution of resource availability in the future should be predicted. Thus, jobs can be scheduled onto the resource nodes with long uptime instead of upcoming failing nodes. Therefore, modeling and predicting the resource availability in the dynamic Grid environments are significant and imperative.

In this paper, we propose a dependable Grid workflow scheduling system (DGWS), which adopts a Markov Chain-based resource availability prediction model. Based on the model, a reliability cost driven workflow scheduling algorithm is presented. The rationale is that it first predicts the reliability of resource node during task execution and then makes scheduling decision in terms of the reliability cost of success execution of task. Performance evaluation is conducted to compare DGWS with three popular workflow scheduling algorithms: HEFT [1], PRMS [5], and eFRCD [6]. Performance evaluation results, including simulation on both parametric randomly generated DAGs and two real scientific workflow applications, demonstrate that DGWS improves the success ratio of tasks and diminishes the makespan of workflow, so improves the dependability of workflow execution.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces the reliability analysis of DAG. The system architecture is designed in Section 4. Then, the reliability cost driven workflow scheduling algorithm is proposed in Section 5. Section 6 contains a brief overview of three frequently used workflow scheduling algorithms that we apply for performance comparison. Section 7 conducts a comparison study of DGWS with the above-mentioned algorithms. Finally, we conclude and give some future work in Section 8.

2 Related Work

Grid workflow researches have attracted more attentions [7, 8], including resources monitoring, service analysis, scheduling algorithms, fault tolerance, and so on. Current workflow scheduling algorithms can be classified into two main groups: heuristic-based and guided random-search-based algorithms [1]. The former can be further classified into three groups: list scheduling heuristics, clustering heuristics and task duplication heuristics. The list-scheduling heuristics are generally more practical and provide better performance results at a lower scheduling time than other groups, so our work is based on the list scheduling. Its representative algorithms are HEFT [1], CPOP [1] and DLS [3].

The workflow scheduling strategies can be categorized into performance-driven, market-driven and trust-driven [9]. The performance-driven strategy tries to submit jobs onto resources to achieve optimal performance for users and system [10]. The work in [11] utilizes the market-driven strategy. In the system, bids are collected from eligible resource providers for each task. If the execution time satisfies user's requirement, a bid with lower price will be chosen as the optimal bid. The reference [12] proposes a hybrid market approach to manage Grid resources: combining futures and spot markets, in which if users are to correctly express their valuations for service, quality of service guarantees would be given with respect to the turnaround time of their workloads. In [13], a new instantiation of the negotiation protocol between the scheduler and resource manager using a market-based Continuous Double Auction (CDA) model is presented to schedule scientific applications in distributed Grid and cloud environments.

In [14], the trust-driven scheduling strategy is adopted to map jobs onto appropriate resources according to their trust levels. The strategy avoids selecting malicious and non-reputable resources so as to increase the system reliability. However, it doesn't consider the job completion time. The references [5, 6] exploit "*Reliability Cost*" as the scheduling objective to improve the reliability of task execution. The "*Reliability Cost*" is defined to be the product of processor failure rate and task

execution time. The algorithms in [5] (MCMS and PRMS) are both based on ALAP (As Late As Possible) scheduling which is proved poorer than the list scheduling [1]. As exploiting the primary-backup fault-tolerance, eFRCD [6] occupies so more precious resources that the job waiting time would become long and many jobs may fail in high system load. In addition, in above reliability cost model, the failure rate of resources is set experimentally.

More attention has been paid to modeling the characteristics of resource availability [15–19]. The references [16, 17] conclude that the time between reboots of nodes is best modeled by a Weibull distribution with shape parameters of less than 1, implying that a node becomes more dependable the longer it has been operating. The reference [19] finds that the resource availability follows exponential, Weibull and Pareto distributions with different parameters. In [20], much work analyzes the machine availability in enterprise systems, but the results are only meaningful for the considered application domain. The references [21, 22] propose a multi-state availability model based on semi-Markov to predict resource availability. However, the model applies to the fine-grained CPU cycle availability prediction and the applications are confined to be CPU-bound batch programs, which are sequential or comprise multiple tasks with little or no inter task communication. Different from the reference [21], in our work, different historical TTFs (Time To Failure) of resources are adopted as the Markov state set to predict the resource availability (TTF) in a future time window.

3 Reliability Analysis of DAG

Directed Acyclic Graph (DAG) is an efficient model to represent workflow application [23]. A DAG G = <V,E,W> is a node-weighted and edge-weighted directed graph, where V = $<n_1,n_2,\ldots,n_n>$ is the set of task nodes, with each node denoting a task, E ⊆ V×V is the weighted edge set that defines the precedence relations among nodes in V. The weight on each edge, $D_{ij} \in W$, denotes the volume of data being transmitted from task node n_i to task node n_j. $P = \{P_1,P_2,\ldots,P_M\}$ represents the resources of a Grid system. For each task $n_i \in V$, the weight on each node, $T(n_i)$, represents the execution time on each resource node: $T(n_i) = \{t_1(i),t_2(i),\ldots,t_M(i)\}$, where $t_j(i)$ represents the execution time of n_i on P_j and can be obtained by the prediction model of GHS proposed in [24]. c_{ij} denotes the communication cost from P_i to P_j, namely network bandwidth.

Consider a Grid system with M resource nodes, $P = \{P_1,P_2,\ldots,P_M\}$, and a DAG containing N task nodes, $V = <n_1,n_2,\ldots,n_n>$. Let x_{ij} be a binary number that denotes whether task n_i is assigned to P_j, 1, for assigned, 0, for not assigned. Let ps_{ij} be the probability of resource node P_j not to fail during the running of task n_i on P_j. Researches have show that the system reliability follows the negative exponential distribution [25, 26]. So, the probability of system not to fail can be expressed in Formula (1).

$$\Pr = \xi g \prod_{j=1}^{M} \prod_{i=1}^{N} \left(ps_{ij}^{x_{ij}\cdot(t_j(i)+T_{lat})} \right)$$

$$T_{lat} = \sum_{p \in prec(i)} \sum_{k=1}^{M} \left(D_{pi} \cdot x_{pk} \cdot c_{kj} \right) + SL_j \tag{1}$$

As there are errors in deriving Pr due to the fluctuating of network bandwidth and nodes' performance, ξ is the fixup parameter and is used to amend the theoretical result of Pr. ξ is usually set empirically (0.7–1.2). T_{lat} denotes the execution latency of task n_i, including the time that task n_i spends to fetch the needed data from its preceded nodes and the scheduling length of resource node P_j (SL_j). prec(i) is the set of immediate precursors of task i. When ps_{ij} is not 0,

$$\Pr \approx \xi g \prod_{j=1}^{M} \prod_{i=1}^{N} \left(e^{-(1-ps_{ij})\cdot x_{ij}\cdot(t_j(i)+T_{lat})} \right) \tag{2}$$

In order to maximize Pr, we need to minimize:

$$\min \left(\sum_{j=1}^{M} \sum_{i=1}^{N} \left(1 - ps_{ij}\right) \cdot x_{ij} \cdot \left(t_j(i)\right) T_{lat} \right) \tag{3}$$

As the reference [6], the *Reliability Cost* (RC) is defined to be the product of resource failure rate and the execution time of task as follows:

$$RC_{ij} = \left(1 - ps_{ij}\right) \cdot \left(t_j\left(i\right) + T_{lat}\right) \tag{4}$$

$$\min\left(\sum_{j=1}^{M}\sum_{i=1}^{N} x_{ij} \cdot RC_{ij}\right) \tag{5}$$

Thus, the Formula (3) can be expressed as Formula (5). From Formulas (2), (3) and (4), we can see that to improve the dependability of workflow execution, we need to minimize RC. The lower RC is, the higher the dependability is. The RC of workflow DAG V on Grid system P is obtained as follows:

$$RC = \sum_{i=1}^{N}\sum_{j=1}^{M}\left(x_{ij} \cdot RC_{ij}\right) \tag{6}$$

4 System Architecture

The DGWS architecture is shown in Fig. 1. The system consists of two main components: *Scheduler* and *Predictor*. DGWS adapts the *Scheduler* to the dynamic Grid environment via collaboration with the *Predictor*.

4.1 Scheduler

The *Scheduler* consists of two components: *Instance* and *Execution Manager*. For each workflow application, an *Instance* is instantiated and first analyzes the relationship of tasks in DAG, and then consults the *Predictor* to estimate the communication and computation cost with the given resource set and calculate the reliability cost of tasks execution on each resource. Finally, the *Instance* decides the resource mapping with the goal of achieving the minimum reliability cost for entire workflow while meeting the QoS requirements, and submits the schedule to the *Execution Manager*. The *Execution Manager* receives the DAG and executes it. It is responsible for getting job input file ready and executing the job on the mapped resource.

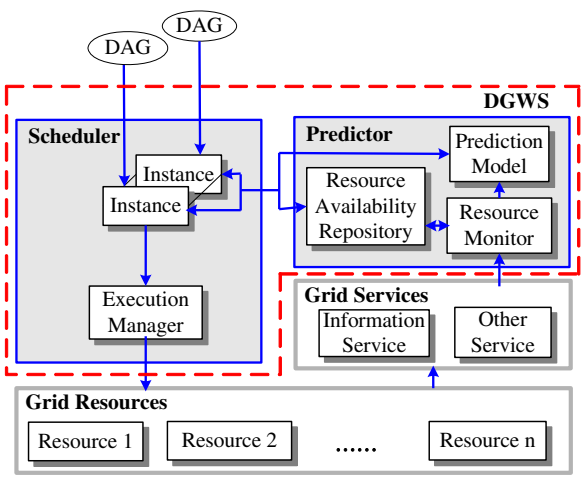


Fig. 1 System architecture

4.2 Predictor

The *Predictor* can be further decomposed into *Prediction Model*, *Resource Availability Repository* and *Resource Monitor*. The *Prediction Model* is based on Markov Chain and can dynamically predict the resource availability and will be detailed in Section 4.3. The *Resource Availability Repository* is used to keep the information of resources, the state space and state transition matrix which are used by the *Prediction Model*. The *Resource Monitor* is used to monitor the change of Grid resources, such as joining and leaving. It triggers the *Prediction Model* while resources join, leave or fail.

4.3 Prediction Model

The Markov model is usually utilized to model the stochastic processes in many fields. Discrete-time Markov Chain (DTMC) is a process that consists of a finite number of states $M(S_1, S_2, \ldots, S_m)$ and $M \times M$ known state transition matrix Q. In matrix Q, Q_{ij} is the probability of moving from state S_i to state S_j [27]. Suppose at time k, system state is S_i $(1 \le i \le M)$ and the distribution of S_i is $Q_k(S_i) = e_i$, where e_i is $1 \times M$ row vector, the value at location i is 1, and others is 0. Thus, we can predict the distribution of S_i at time k+1: $Q_{k+1}(S_i) = Q_k(S_i) \cdot Q = e_i \cdot Q$. At time k+2, the distribution of S_i is: $Q_{k+2}(S_i) = Q_{k+1}(S_i) \cdot Q = e_i \cdot Q^2$. At time k+n, the distribution of S_i is: $Q_{k+n}(S_i) = Q_{k+n-1}(S_i) \cdot Q = e_i \cdot Q^n$. So, with DTMC, we can

get the occurrence probability of each state at each time.

The TTF (Time To Failure) denotes the running time of a system before failure occurs and is usually adopted as a basic metric for evaluating system availability. So, the paper uses TTF to represent the resource availability. In the Grid environment, resources are volatile and failures can occur at any time. Therefore, Markov Chain model can be used to model the stochastic process of resources' TTF. In the prediction model, TTF is adopted as system state. In traditional Markov model, the M and Q are invariable. However, in dynamic Grid environments, frequent resource failures can generate amounts of TTF, so requiring large storage space for M and Q, which makes the model complex and unpractical for Grid. To address the issue, we present an adaptive Markov Chain-based Grid node TTF prediction model which can dynamically amend M and Q.

When a resource node fails, a new TTF is produced (denoted by TTF_{new}). Then, the status space M would be traversed. If there exists state S_i whose absolute difference value and TTF_{new} is less than the specified value, S_i would be modified to be the average of S_i and TTF_{new} and the number of state transition would be increased by 1. Reversely, if there doesn't exist this state, new state S_{m+1} would be created. At the same time, Q would be emended according to Formula (7).

$$Q_{ij} = tn_{ij} / N_{total},\ N_{total} = \sum_{K} tn_{ik} \tag{7}$$

Where tn_{ij} represents the transition number from state i to state j at K failures. N_{total} denotes the all state transition number of K failures.

To better understand how to create the Markov based prediction model and the transiting process of M and Q, Fig. 2 is took as an example. In Fig. 2, resource node experiences 4 failures and generates 3 different TTFs, namely 3 states. The transiting process of M and Q is shown as follows.

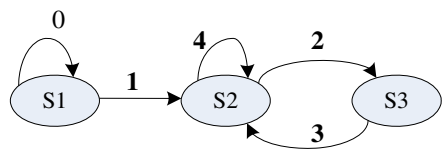


Fig. 2 Transiting of resource nodes' status

Step 0: the resource node starts and no failure occurs. System is at state S_1(S_1 can be set empirically), Q is:

$$Q_0 = [1]$$

Step 1: When the first failure occurs, a new TTF is produced. Then, system has two states: S_1 and S_2, $M(S_1, S_2)$, Q is:

$$Q_1 = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 1 \end{bmatrix}$$

Step 2: When the second failure occurs, a new TTF is produced. Then, system has three states: S_1, S_2 and S_3, $M(S_1, S_2, S_3)$, Q is:

$$Q_2 = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 3: When the third failure occurs, system state transits from S_3 to S_2. Then, system state space M is $M(S_1, S_2, S_3)$, Q is:

$$Q_3 = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 1/2 & 1/2 \end{bmatrix}$$

Step 4: When the forth failure occurs, system state stays at S_2. Then, system state space M is $M(S_1, S_2, S_3)$, Q is:

$$Q_4 = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 2/3 & 1/3 \\ 0 & 1/2 & 1/2 \end{bmatrix}$$

Through the transiting of M and amending of Q, the occurrence probability of each state in the future can be predicted.

5 DGWS Algorithm

5.1 Outline

The DGWS scheduling algorithm is based on the list scheduling and consists of three phases: ranking, grouping and scheduling. Firstly, a weight is assigned to each node and edge of DAG; this is based on averaging all possible values of the cost of node (or edge, respectively) on each resource (or combination of resources respectively). With this weight, upward rank value is computed and each node of DAG is assigned a rank value. The upward rank value of node i, $\mathrm{rank_u(n_i)}$, is recursively defined according to Formula (8) as the reference [1]:

$$rank_u(n_i) = \overline{w_i} + \max_{n_k \in succ(n_i)} (\overline{c} \cdot D_{ik} + rank_u(n_k))$$

$$\overline{w_i} = \sum_{j=1}^{M} t_j(i) \Big/ M \quad \overline{c} = \sum_{i=1}^{M} \sum_{j=1}^{M} (c_{ij}) \Big/ (2 \cdot C_M^2)$$

$$C_M^2 = \frac{M!}{M!\,(M-2)!}. \tag{8}$$

Where $\overline{w_i}$ is the average weight of task node i, $succ(n_i)$ is the set of immediate successors of task node i and $\overline{c}$ is the average communication cost between any two nodes.

Secondly, the task nodes of DAG are sorted in descending order of $\mathrm{rank_u(n_i)}$. With this order, they are divided into different groups as follows. The first node is added to a group numbered 0. If the successive nodes in descending order of their upward rank value are independent with all nodes already assigned to the group (namely, there is no dependence between them), they are placed in the same group. Reversely, if there is dependence, a new group will be created and the new group's number is the current group's number increased by one, and then the node with the smallest rank value is the member of new group. Again, the subsequent task nodes will be assigned to different groups. The final outcome is a set of ordered groups.

Thirdly, according to the ascending order of groups' number, the independent tasks within each group are scheduled. In DGWS, the independent tasks in each group are scheduled based on the reliability cost of the success execution of task on the given resource. The reliability cost driven independent task scheduling algorithm will be detailed in Section 5.2.

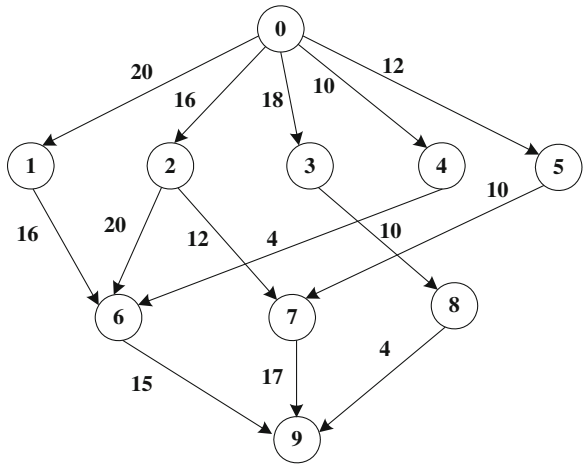


Fig. 3 A sample DAG

In order to illustrate the idea of DGWS algorithm, consider the sample DAG shown in Fig. 3. The number next to each edge of the graph corresponds to the amount of data that needs to be passed from a task to an immediate successor. The cost to execute each of tasks in the graph on each of three resources is given in Table 1. Table 2 shows the cost to transfer a data unit for any given combination of resources. Thus, the cost to transfer, for instance, the data needed from task 0 to task 1 would be 20×2.0 if one of the tasks is executed by resource node 0 and the other by resource node 1.

Table 1 The computation cost

	Nodes		
Task	m0	m1	m2
0	20	22	26
1	24	20	22
2	20	14	18
3	10	8	6
4	21	18	19
5	26	22	24
6	20	19	22
7	52	56	51
8	18	19	17
9	22	20	18

Table 2 The communication cost for the resource nodes

Nodes	Communication cost for a data unit
m0-m1	2.0
m0-m2	1.5
m1-m2	2.5

Table 4 Partitioning the nodes into groups according to their upward rank values

Group	Tasks
0	{0}
1	{5, 2, 1}
2	{7, 4, 3}
3	{6, 8}
4	{9}

The first phase involves the assignment of weights to the nodes and edges of the graph and the computation of upward ranking for the nodes. The results are shown in Table 3. The nodes in descending order of their ranking value are $\{n_0, n_5, n_2, n_1, n_7, n_4, n_3, n_6, n_8, n_9\}$. The second phase involves the partitioning of nodes into ordered groups, considering them in descending order of their upward rank value. Node 0 is assigned to group 0. Node 5 can't be in the same group as node 0 (since it depends on node 0), and, as a result, a new group (group 1) is created. Nodes 1 and 2 can also be in the same group as node 5 (that is group 1, since all three nodes are independent). Node 7 depends on nodes 5 and 2, therefore a new group needs to be created, and so on. When this procedure is completed, nodes are grouped in 5 groups as shown in Table 4. Within each group, the tasks are independent and can be scheduled and run in parallel.

Although the DGWS algorithm seems similar to the hybrid heuristic in [1, 28], there is a fundamental difference. The hybrid heuristic in [1, 28] aims to minimize the makespan of DAG while scheduling independent tasks without considering the resource reliability. So, even if the tasks are scheduled to the resource nodes with minimum makespan, the rescheduling can lead to large makespan due to the resource failures. Whereas, DGWS targets to minimize the reliability cost so as to improve the dependability of success execution of DAG. Thus, the reliable execution can avoid task failure and rescheduling, and lower the makespan of task.

Table 3 Upward ranking of nodes using mean values to compute weights

Task(n_i)	Weight	$Rank_u(n_i)$
0	22.67	203
1	22	124.33
2	17.33	148.33
3	8	74
4	19.33	97.66
5	24	151
6	20.33	70.33
7	53	107
8	18	46
9	20	20

5.2 Reliability Cost Driven Independent Tasks Scheduling

Existing heuristics are developed at the assumption that resources are dedicated and no failures occur, without consideration of resource reliability. Inspired by the reliability analysis of DAG in Section 3 and based on the resource availability prediction, the paper exploits a reliability cost driven independent task scheduling algorithm in DGWS.

The reliability cost driven scheduling algorithm consists of two steps. At the first step, the independent tasks are sorted in descending order of their average completion time on all resource nodes. The average completion time of task i ($ACT(n_i)$) includes the execution time and transfer time of needed data as shown in Formula (9). $prec(n_i)$ is the set of immediate precursors of task node n_i.

$$ACT(n_i) = \overline{w_i} + \sum_{k \in prec(n_i)} (\overline{c} \cdot D_{ki}) \tag{9}$$

At the second step, according to the descending order of ACT, the task with the highest value is selected. The completion time of task i on resource node j (CT_{ij}) is computed according to Formula (10).

$$CT_{ij} = t_j(i) + \sum_{p \in prec(i)} \sum_{k=1}^{M} (D_{pi} \cdot x_{pk} \cdot c_{kj}) \tag{10}$$

Then, using the prediction model in Section 4.3, the reliability of resource node j during the

execution of task i is predicted as follows. Assuming that the state of resource node j is S_q at time t_q, we can predict its state distribution at time t_{q+1}: (Q_{q1} Q_{q2} … Q_{qM}). $S_k(1 \le k \le M)$ denotes the TTF of resource node j. In order to guarantee the dependable execution of task i on resource node j, the following condition should be satisfied: $S_k - (T_{now} - T_j) > CT_{ij}$. Here, T_{now} denotes the system current time and ST_{ij} represents the startup time of resource node j. So, the reliability of success execution of task i on resource node j (namely, ps_{ij}) can be obtained according to Formula (11).

$$ps_{ij} = \sum_{k=x}^{M} (S_k \cdot Q_{qk}) \Big/ \sum_{h=1}^{M} (S_h \cdot Q_{qk})$$
$$S_k - (T_{now} - ST_j) > CT_{ij} \tag{11}$$

Accordingly, the reliability cost of success execution of task i on resource node j, RC_{ij}, can be computed in terms of Formula (4) and the result is shown as follows.

$$RC_{ij} = (1 - ps_{ij}) \cdot (CT_{ij} + SL_{ij}) \tag{12}$$

Thus, we chose the resource node on which the reliability cost of success execution of task is lowest and schedule the task to it. In this way, we can minimize the reliability cost of workflow and improve the dependability of workflow execution.

6 Other Heuristic Scheduling Algorithms for DAG

This section reviews three frequently used DAG scheduling algorithms: HEFT [1], PRMS [5] and eFRCD [6], which are used to conduct the performance evaluation in the coming section.

6.1 HEFT

The HEFT [1] covers two major phases. In task prioritizing phase, it computes the priorities of all tasks in terms of upward rank value. In processor selection phase, it selects tasks in the order of their priorities and schedules each selected task to the resource which minimizes its earliest finish time. Figure 4 shows the schedule obtained by HEFT for the sample DAG of Fig. 3. The scheduling

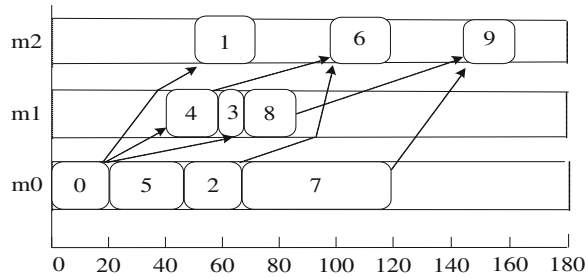


Fig. 4 HEFT Schedule of tasks of DAG in Fig. 3 (schedule length = 161.5)

order of the tasks is {n_0,n_5,n_2,n_1,n_7,n_4,n_3,n_6,n_8,n_9} and the schedule length is equal to 161.5.

6.2 PRMS

The PRMS [5] progressively improves the reliability based on the schedule obtained by the ALAP (As Late As Possible) scheduling. PRMS first schedule all tasks of DAG using the ALAP scheduling. Based on the schedule, PRMS repeatedly take the task with earliest starting time and reschedule it to a resource such that the system reliability cost is minimized. Meantime, its finish time must be earlier than that in ALAP and the task doesn't overlap with other tasks remaining in the ALAP schedule. In PRMS, the key issue is obtaining the critical path. The tasks with the same summation of upward and downward ranks compose the critical path of a DAG. The downward rank values of tasks are obtained in terms of Formula (13) as in the reference [1].

$$rank_d(n_i) = \max_{n_k \in prec(n_i)} (rank_d(n_k) + \overline{w_k} + \overline{c} \cdot D_{ki}) \tag{13}$$

As shown in Table 5, the tasks {n_0,n_2,n_7,n_9} are the critical tasks. Then, according to the deadline of DAG and the critical path, we can obtain the EST (Earliest Starting Time) and LST (Latest Starting Time), and conduct the ALAP scheduling.

6.3 eFRCD

The eFRCD [6] uses a Primary/Backup scheduling technique to tolerate the single processor failure. Meanwhile, it adopts the reliability cost as

Table 5 Upward and downward rank value of task nodes using mean computation cost and communication cost

Task(n_i)	$Rank_u(n_i)$	$Rank_d(n_i)$	$Rank_u(n_i)+rank_d(n_i)$
0	203	0	203
1	124.33	62.67	187
2	148.33	54.67	203
3	74	58.67	132.67
4	97.66	42.67	140.33
5	151	46.67	197.67
6	70.33	116.67	187
7	107	96	203
8	46	86.67	132.67
9	20	183	203

scheduling objective to improve the reliability. The eFRCD schedules tasks in the following three main steps. First, tasks are ordered by their deadlines in non-decreasing order, such that tasks with tighter deadlines have higher priorities. Second, the primary copies are scheduled. Finally, the backup copies are scheduled in a similar manner as the primary copies and the backup copies of tasks are allowed to be overlapped.

7 Performance Evaluation

The prototype of DGWS is implemented and integrated into CGSP (ChinaGrid Support Platform) [29]. We compare DGWS with HEFT [1], PRMS [5] and eFRCD [6]. The testbed employs a cluster deployed with CGSP with 16 1.3 GHz IA 64 nodes (each with 2 GB memory), which are connected by a 100 Mbps Ethernet switch and run Redhat Linux 9.0. In addition, GridSim is also adopted to simulate the Grid environment [30] on which the simulation experiments are conduced. However, due to the length of paper, the simulation experimental results are omitted.

7.1 Evaluation Metrics

(1) Reliability Cost (RC). RC is defined to be the product of resource failure rate and the execution time of tasks of a DAG as shown in Formula (6).

(2) Success ratio. It is defined as the ratio of the number of success jobs to the number of all jobs.

(3) Makespan. It is the time for obtaining the output result of a given DAG, including the running time of scheduling algorithm and the execution time of tasks of DAG.

7.2 Results of Parametric Randomly Generated DAGs

7.2.1 Random DAG Generator

In our study, a random DAG generator is implemented to generate weighted DAGs with various characteristics that depend on several input parameters given below.

(1) Number of tasks in DAG: V.

(2) The max and min weights of task nodes of DAG: DAGNode_Max_Weight, DAGNode_Min_Weight. The average computation cost of each task (n_i) in DAG ($\overline{w_i}$) is selected randomly from a uniform distribution with range [DAGNode_Max_Weight, DAGNode_Min_Weight].

(3) Communication to computation ratio: CCR ($0<CCR<1$). It is the ratio of the average communication cost to the average computation cost. If a DAG's CCR value is very low, it can be considered as a computation-intensive application. Otherwise, it is a data-intensive application. The volume of data being transmitted from task n_i to task n_j in DAG (D_{ij}) is selected randomly from a uniform distribution with range $CCR \times$[DAGNode_Max_Weight, DAGNode_Min_Weight] $\cup\{0\}$.

(4) Offset ratio of computation costs on resource nodes: ε ($0 \leq \varepsilon < 1$). It is basically the heterogeneity factor to represent the heterogeneous features of resources. A high offset ratio indicates a significant difference in a task's computation cost among the resource nodes and a low offset ratio shows that the execution time of a task is almost equal on any given Grid resource. The computation cost of task n_i of DAG on resource node P_j is selected randomly from a uniform distri-

bution with range $[\overline{w_i} \cdot (1-\varepsilon), \overline{w_i} \cdot (1+\varepsilon)]$. While the value of ε is zero, the computation cost of task n_i on any resource node is $\overline{w_i}$, representing the target resources are homogeneous.

(5) Adjustment factor of the number of nodes in each level of DAG: ϕ $(0 < \phi < 1)$. The number of nodes in ith level of DAG (NL_i) is selected randomly from a uniform distribution with range $[1, \phi \cdot (\text{V-2-NL}_{i-1})]$.

To introduce the scheduling model clearly, the first and the last level of the generated DAG contain only one node. In practical, this is not a real restriction. Thus, to generate a DAG with a number of nodes, we first generate a single entry node and a single exit node, and then other nodes are divided into levels. Care is taken so that the entry node in DAG is connected to all the nodes of the 2nd level and the exit node is connected to all the nodes of the penultimate level. To introduce the scheduling model clearly, Each node in ith level (2<i<L−2) is connected to m nodes of (i + 1)th level. Here, L is the total number of levels of DAG and m is selected randomly from a uniform distribution with range $[1, NL_{i+1}]$.

7.2.2 Failure Simulator

Due to the factors of security, business secret and etc., the system running logs of most enterprises cannot be obtained. Since the failure traces from real large-scale systems are unavailable, plus failures in distributed systems are correlated temporally and spatially, not identically distributed [15, 18, 31], we implement one failure simulator according to the reference [31] to model the failure distribution including the occurrence time, location distribution and downtime of failures. The failure simulator utilizes the Weibull distribution to model the occurrence time of failures and the Zipf's law to the location distribution of failures. The Weibull distribution function (denoted by $F(t)$) can be described in Formula (14). The parameter α is called the shape parameter, and η is the scale parameter.

$$F(t) = 1 - e^{-(t/\eta)^{\alpha}} \tag{14}$$

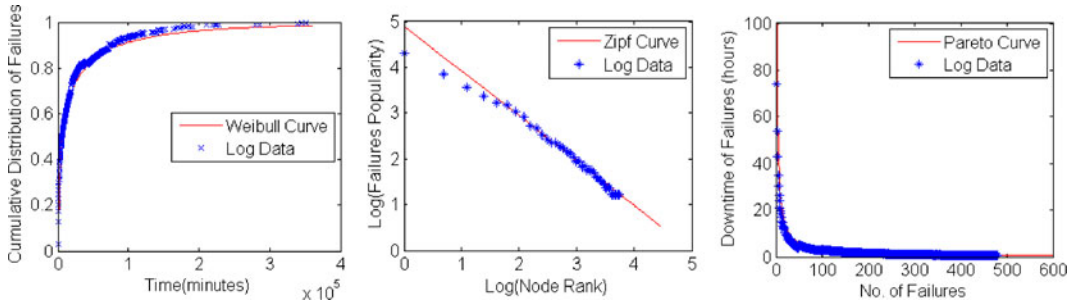
The Zipf's law can be given in Formula (15). The β reflects the degree of popularity skew, while K represents the number of failures for the most volatile nodes.

$$P(i) = K/i^{\beta}, \quad \beta \in [0.5, 1] \tag{15}$$

We exploit the Pareto distribution to model the downtime of failures, because hardware manufacturing technology continues to improve so that the number of permanent faults is gradually decreasing and the transient faults increasingly become the main reason for resource failures. The reference [32] makes detailed research on the historical logs of IT systems of IBM and Los Angeles national laboratory, and shows that the downtime of system follows heavy-tailed distribution and the Pareto distribution can be used to efficiently model the downtime of resource nodes. The Pareto distribution function (F(x)) is given in Formula (16). Here, λ is the heavy-tailed parameter determining the degree of heavy-tailed distribution. The parameter θ denotes the tail start point of heavy-tailed distribution.

$$F(x) = 1 - (\theta/x)^{\lambda} \quad \lambda, \theta > 0, x \geq \theta \tag{16}$$

To validate the accuracy and efficiency of the failure simulator, we use one data set which exhibits failure behavior typical of Grid resources currently residing on the Wuhan Grid Center of ChinaGrid [29]. The site is comprised of 38 nodes and is at Cluster and Grid Computing Lab (CGCL) at Wuhan, China. The data set is the historical logs of resource failures from July 2006 to July 2007. For each above theoretical distribution, we exploit two methods to compute the needed parameters to match the distribution to the empirical data. One is the Maximum Likelihood Estimation (MLE) based on Matlab scripts [33]. Another is EMpht software which is suitable to deal with large data sets [34]. From the Fig. 5, we can conclude that the failure simulator can model the resource availability efficiently with proper parameters.



(a) η=13200, α=0.44 **(b) K=135, β=0.98** **(c) θ=140, λ=0.77**

Fig. 5 Failures distribution of CGCL. **a** Cumulative distribution of failures vs. Weibull distribution, **b** The logarithm of failures popularity vs. Zipf's distribution, and **c** Downtime of failures vs. Pareto distribution

7.2.3 Efficiency of Resource Availability Prediction Model

To evaluate the Markov Chain based resource availability prediction model, based on the trace data set in Section 7.2.2, we construct the Markov Chain prediction model and predict the resource availability.

The data set is the historical log of resource failures of 5 nodes at CGCL from July 2006 to February 2007. TTFs of nodes are shown in Table 6.

To predict the resource availability, the Markov Chain prediction model must be constructed in advance. First, we construct the state space M and state transition matrix Q using the 80 % of TTFs. Then, based on the M and Q, the rest 20 % of TTFs are predicted. Finally, through comparing the predicted TTFs and the real TTFs, we can obtain the performance evaluation of prediction model. The accuracy rates of resource availability prediction are shown in Table 7. From the Table 7, we can see that the accuracy rate of node 1 is highest than other nodes. The reason is that the prediction model works at the base of historical logs of nodes. So, if there are more data of TTFs, the state space M and state transition matrix Q would be generated perfectly and predict precisely.

7.2.4 Simulation Results

In all experiments, the input parameters are restricted to the following values:

V∈{20, 40, 60, 80, 100},
DAGNode_Max_Weight (sec) ∈ {100, 200, 300},
DAGNode_Min_Weight (sec) ∈ {20, 40, 80},
CCR ∈ {0.5, 1.0, 4.0},
$\varepsilon \in \{0.2, 0.8\}$,
$\phi \in \{0.3, 0.6, 1\}$.

These combinations produce 810 different DAG types. Since 20 random DAGs are generated for each DAG type, the total number of DAGs is 16,200. We simulate the resource failures using failure injection mechanism. We fix the scale parameter of the Weibull distribution and by varying the value of shape parameter (α), obtain different number of failures. The failure occurrence time distributions are shown in Table 8. In Formula (15), K represents the number of failures of the most volatile nodes and β reflects the degree of popularity skew. We set $K = 46$

Table 6 Number of TTFs of nodes

	Nodes				
	1	2	3	4	5
Number of TTF	167	141	112	109	78

Table 7 Accuracy rates of node availability prediction

	Nodes				
	1	2	3	4	5
Accuracy rates (%)	90.24	88.56	86.37	87.02	86.84

Table 8 Failures occurrence time distribution with different shape (α) values

Scale η	Shape α	Number of failures	Failures/minute
18000	0.083	106	0.016
	0.871	3135	2.177

and $\beta = 0.76$ according to the real environments. As we know that the higher β corresponds to a highly skewed distribution in which the majority of failures are concentrated on a relatively small number of nodes. In addition, $\theta = 127$, $\lambda = 0.87$. The higher λ implies that the downtime of most failures is shorter.

For the sake of brevity, the paper only shows the experiment results of two situations: very low failure ratio and high failure ratio. In addition, as the short deadline constraint results in poorer performance in schedulability so that many tasks are rejected, which are also proved in [5, 6], only the results with large deadline constraint of DAG tasks are shown in the paper. For the failed jobs, DGWS reschedules them to other nodes based on reliability cost.

Figures 6 and 7 show the average performance values for each set of DAGs. When the failure ratio is very low, HEFT and DGWS perform better and eFRCD is the poorest. Whereas, HEFT is the poorest and DGWS performs best at average makespan in high failure ratio. The reasons are as follow: (1) HEFT aims to schedule job to the resource with minimum earliest finish time without consideration of resource reliability. Therefore, in stable environment, for DGWS, PRMS and eFRCD, computing reliability cost would incur extra overhead, which increases the makespan of jobs. In unstable environment, frequent resource failures lead to the low job success ratio of HEFT and rescheduling the failed jobs would augment the makespan; (2) As exploiting the primary-backup fault tolerance and reliability cost, eFRCD performs better in low system load. In high system load, the primary-backup policy would occupy so more resources that the job waiting time becomes long and many jobs may fail. Most of all, eFRCD is designed to tolerate a single resource failure. For the simultaneous multiple resource failures, however, eFRCD may incur extra load and lead to poor performance; (3) PRMS is based on the ALAP (As Late As Possible) scheduling which is proved poorer than the list scheduling [1]; (4) DGWS considers the resource reliability. Moreover, it integrates the advantages of the list scheduling and group scheduling.

7.3 Results of Real Scientific Workflow Applications

Finally, we consider the application graphs of two real scientific workflow problems. One is a computation-intensive application (Gaussian Elimination [35]) and the other is a data-intensive application (BLAST [36]). We construct the resource availability prediction model based on the traces of ChinaGrid resources in CGCL from August 2007 to November 2007. The experiments are conducted respectively in different time periods: daytime, nighttime, workday, and weekend.

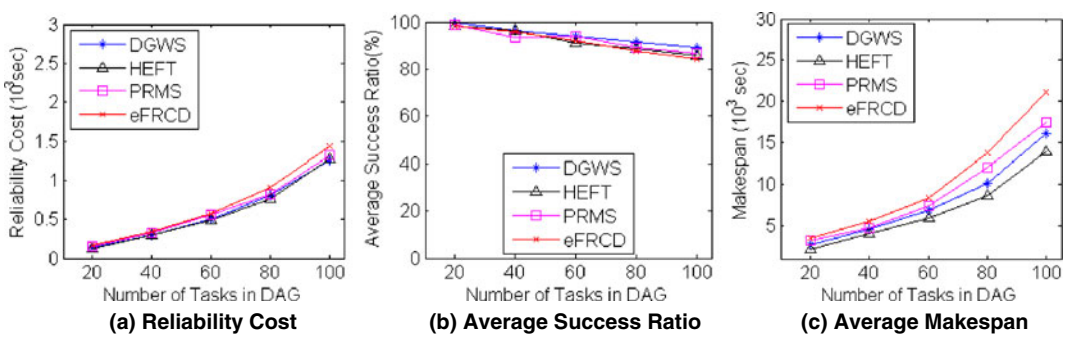


Fig. 6 Performance results at failures distribution with $\eta = 18000$, $\alpha = 0.083$, $K = 46$, $\beta = 0.76$, $\theta = 127$, $\lambda = 0.87$

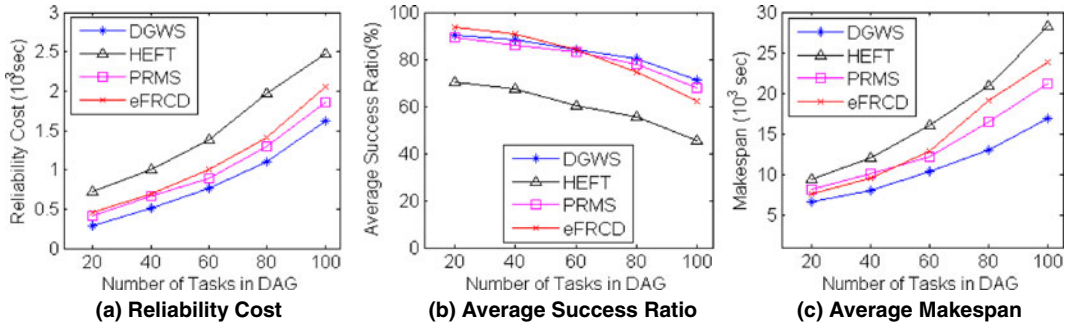


Fig. 7 Performance results at failures distribution with $\eta = 18000$, $\alpha = 0.871$, $K = 46$, $\beta = 0.76$, $\theta = 127$, $\lambda = 0.87$

Table 9 Failures occurrence time distribution

Scale η	Shape α	Number of failures	Failures/minute
18000	0.32	1428	0.992

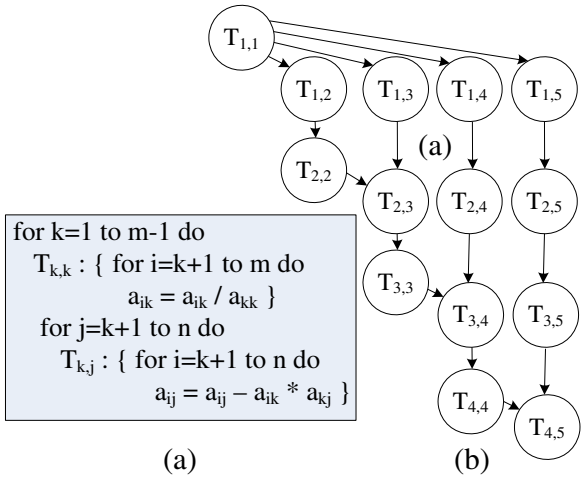


Fig. 8 **a** The Gaussian elimination algorithm. **b** The task graph for a matrix of size 5

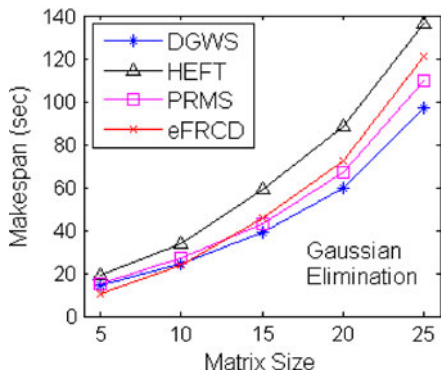


Fig. 9 Average Makespan at failures distribution with $\eta = 18000$, $\alpha = 0.32$, $K = 46$, $\beta = 0.76$, $\theta = 127$, $\lambda = 0.87$

We set $\eta = 18000$, $\alpha = 0.32$, $K = 46$, $\beta = 0.76$, $\theta = 127$, $\lambda = 0.87$. The failure occurrence time distributions are shown in Table 9.

Gaussian Elimination is a method for solving matrix equations of the form Ax=b. In Gaussian Elimination application graph, the number of tasks v, and the number of graph levels l depends on the matrix size m. The total number of tasks v in a Gaussian Elimination graph is equal to $(m^2+m-2)/2$. Figures 8 give the sequential program for the Gaussian elimination algorithm and the data-flow graph of $m = 5$. Figure 9 shows the makespan of 100 generated graphs for each matrix size m ∈ {5,10,15,20,25}.

The BLAST is used to find regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance

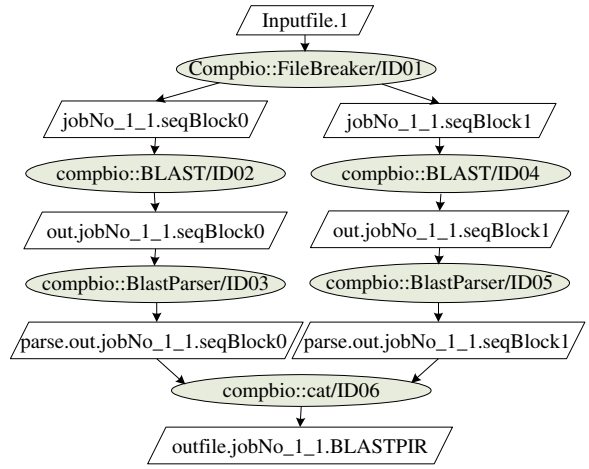


Fig. 10 A six-step BLAST workflow with two-way parallelism [27]

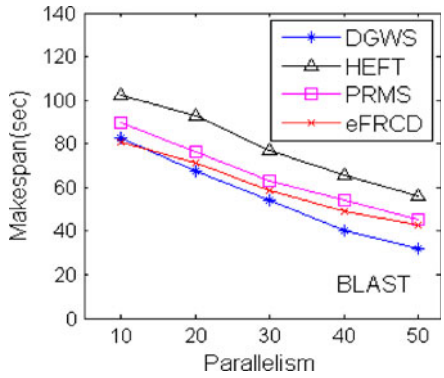


Fig. 11 Average makespan at failures distribution with $\eta = 18000$, $\alpha = 0.32$, $K = 46$, $\beta = 0.76$, $\theta = 127$, $\lambda = 0.87$

of matches. Our experiments adopt a six-step BLAST workflow example with different way parallelism. It produces a simple comparative analysis of protein sequences. Figure 10 gives a six-step BLAST workflow example with two-way parallelism. The ellipse represents a job and the parallelogram represents data file. We conduct the simulation with 10-, 20-, 30-, 40- and 50-way parallelism respectively and the makespan is shown in Fig. 11.

These two real world applications further confirm that DGWS performs better than HEFT, PRMS and eFRCD in the dynamic Grid environments.

8 Conclusion and Future Work

In the paper, the reliability of DAG is firstly analyzed. Then, a dependable Grid workflow scheduling system is presented. It introduces a Markov Chain-based resource availability prediction model. Based on the model, a reliability cost driven workflow scheduling algorithm is proposed, which considers both the task completion time and resource availability. Finally, performance evaluation is conducted and the results are promising. As our future work, we plan to perfect the resource availability prediction model and further research the dependable workflow execution using fault tolerance technology such as checkpoint.

Acknowledgements This paper is supported by Nation Science Foundation of China under No. 61172086, 61170223, 60973037, 61073024 and 612320081 and Outstanding Youth Foundation of Hubei Province under Grant No.2011CDA086.

References

1. Topcuoglu, H., Hariri, S., Wu, M.: Performance effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
2. Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling strategies for mapping application workflows onto the Grid. In: Proc. of 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), pp. 125–134. IEEE Computer Society, Research Triangle Park, North Carolina, USA (2005)
3. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture. IEEE Trans. Parallel Distrib. Syst. **4**(2), 175–187 (1993)
4. Hwang, S., Kesselman, C.: Grid workflow: a flexible failure handling framework for the Grid. In: Proc. of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pp. 126–137, Seattle, Washington, USA. IEEE Computer Society Press, Los Alamitos, CA, USA, (2003)
5. He, Y., Shao, Z., Xiao, B., Zhuge, Q., Sha, E.: Reliability driven task scheduling for heterogeneous systems. In: The 15th IASTED International Conference on Parallel and Distributed Computing and Systems 1, pp. 465–470 (2003)
6. Qin, X., Jiang, H., Swanson, D.R.: An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In: Proc. of the 2002 International Conference on Parallel Processing, pp. 360–368 (2002)
7. Truong, H.L., Fahringer, T., Dustdar, S.: Dynamic instrumentation, performance monitoring and analysis of Grid scientific workflows. J. Grid Computing **3**(1–2), 1–18 (2005)
8. Yu, J., Buyya, R.: A taxonomy of workflow management systems for Grid computing. J. Grid Computing **3**(3–4), 171–200 (2005)
9. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of Grid resource management systems for distributed computing. Softw. Pract. Exp. **32**(2), 135–164 (2002)
10. Cao, J., Jarvis, S.A., Saini, S., Nudd, G.R.: GridFlow: workflow management for Grid computing. In: 3rd International Symposium on Cluster Computing and the Grid (CCGrid). IEEE Computer Society Press, Los Alamitos, Tokyo, Japan (2003)
11. Buyya, R., Murshed, M., Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global

Grids: a deadline and budget constrained cost-time optimization algorithm. Softw. Pract. Exp. (SPE) J. **35**(5), 491–512 (2005)
12. Vanmechelen, K., Depoorter, W., Broeckhove, J.: Combining futures and spot markets: a hybrid market approach to economic Grid resource management. J. Grid Computing **9**(1), 81–94 (2011)
13. Prodan, R., Wieczorek, M., Mohammadi Fard, H.: Double auction-based scheduling of scientific applications in distributed Grid and cloud environments. J. Grid Computing **9**(4), 531–548 (2011)
14. Song, S.S., Hwang, K., Kwok, Y.K.: Trusted Grid computing with security binding and trust integration. J. Grid Comput. **3**(1–2), 53–73 (2005)
15. Sahoo, R., Sivasubramaniam, A., Squillante, M.S., Zhang, Y.: Failure data analysis of a large-scale heterogeneous server environment. In: The International Conference on Dependable Systems and Networks (DSN), Florence, Italy (2004)
16. Heath, T., Martin, R., Nguyen, T.D.: Improving cluster availability using workstation validation. In: The ACM SIGMETRICS 2002, pp. 217–227. Marina Del Rey, CA (2002)
17. Sahoo, R., Oliner, A.J., Rish, I., Gupta, M., Moreira, J.E., Ma, S., Vilalta, R., Sivasubramaniam, A.: Critical event prediction for proactive management in large-scale computing clusters. In: Proc. of the ACM SIGKDD, pp. 426–435 (2003)
18. Fu, S., Xu, C.-Z.: Quantifying temporal and spatial correlation of failure events for proactive management. In: Proc. of IEEE International Symposium on Reliable Distributed Systems (SRDS), pp. 175–184 (2007)
19. Nurmi, D., Brevik, J., Wolski, R.: Modeling machine availability in enterprise and wide-area distributed computing environments. In: Technical Report CS2003–28, U.C. Santa Barbara Computer Science Department (2003)
20. Brevik, J., Nurmi, D., Wolski, R.: Automatic methods for predicting machine availability in desktop Grid and peer-to-peer systems. In: Proc. of CCGrid'04, pp. 190–199 (2004)
21. Ren, X.J., Lee, S., Eigenmann, R., Bagchi, S.: Resource failure prediction in fine-grained cycle sharing systems. In: Proc. of 15th IEEE International Symposium on High Performance Distributed Computing, pp. 93–104. IEEE Computer Society Paris, France (2006)
22. Ren, X.J., Lee, S., Eigenmann, R., Bagchi, S.: Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. J. Grid Computing **5**(2), 173–195 (2007)
23. Malewicz, G., Foster, I., Rosenberg, A.L., Wilde, M.: A tool for prioritizing DAGMan jobs and its evaluation. J. Grid Computing **5**(2), 197–212 (2007)
24. Wu, M., Sun, X.H.: Grid harvest service: a performance system of Grid computing. J. Parallel Distrib. Comput. **66**(10), 1322–1337 (2006)
25. Sen, A., Bhattacharyya, G.K.: A piecewise exponential model for reliability growth and associated inferences. In: Basu, A.P. (ed.) Advances in Reliability, pp. 331–355. Elsevier (1993)
26. Calabria, R., Guida, M., Pulcini, G.: A Bayes procedure for estimation of current system reliability. IEEE Trans. Reliab. **41**, 616–620 (1992)
27. Gilks, W.R., Richardson, S., Spiegelhalter, D.J.: Introducing Markov chain Monte Carlo. In: Gilks, W.R., Richardson, S., Spiegelhalter, D.J. (eds.) Markov Chain Monte Carlo in Practice, pp. 1–19. Chapman & Hall, London (1996)
28. Sakellariou, R., Zhao, H.: A hybrid heuristic for DAG scheduling on heterogeneous systems. In: Proc. of 13th Heterogeneous Computing Workshop (HCW-2004), Santa Fe, New Mexico, USA (2004)
29. Jin, H.: ChinaGrid: making Grid computing a reality. In: Digital Libraries: International Collaboration and Cross-Fertilization, Lecture Notes in Computer Science, vol. 3334, pp. 13–24. Springer (2004)
30. Buyya, R., Murshed, M.: GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. J. Concurr. Comput. Pract. Exp. **14**(13–15), 1175–1220 (2002)
31. Zhang, Y., Squillante, M.S., Sivasubramaniam, A., Sahoo, R.K.: Performance implications of failures in large-scale cluster scheduling. In: 10th Workshop on JSSPP, SIGMETRICS, pp. 233–252 (2004)
32. Kato, S., Osogami, T.: Evaluating availability under quasi-heavy-tailed repair times. In: Proc. of Dependable Systems and Networks with FTCS and DCC, 2008, DSN 2008, pp. 442–451 (2008)
33. Matlab by Mathworks: http://www.matlab.com. Accessed 1 Aug 2011
34. Asmussen, S., Nerman, O., Olsson, M.: Fitting phase-type distributions via the EM algorithm. Scand. J. Statist. **23**, 419–441 (1996)
35. Cosnard, M., Marrakchi, M., Robert, Y., Trystram, D.: Parallel gaussian elimination on an MIMD computer. Parallel Comput. **6**, 275–295 (1988)
36. Sulakhe, D., Rodriguez, A., D'Souza, M., Wilde, M., Nefedova, V., Foster, I., Maltsev, N.: GNARE: an environment for Grid-based high throughput genome analysis. In: Proc. of 5th IEEE Int. Symp. Cluster Computing and Grid (CCGrid05), vol. 1, pp. 455–462. Cardiff, UK (2005)