J Grid Computing (2012) 10:325–346
DOI 10.1007/s10723-012-9215-6

Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid

Adán Hirales-Carbajal · Andrei Tchernykh · Ramin Yahyapour · José Luis González-García · Thomas Röblitz · Juan Manuel Ramírez-Alcaraz

Received: 22 December 2011 / Accepted: 13 March 2012 / Published online: 29 March 2012

Abstract In this paper, we present an experimental study of deterministic non-preemptive multiple workflow scheduling strategies on a Grid. We distinguish twenty five strategies depending on the type and amount of information they require. We analyze scheduling strategies that consist of two and four stages: labeling, adaptive allocation, prioritization, and parallel machine scheduling. We apply these strategies in the context of executing the Cybershake, Epigenomics, Genome, Inspiral, LIGO, Montage, and SIPHT workflows applications. In order to provide performance comparison, we performed a joint analysis considering three metrics. A case study is given and corresponding results indicate that well known DAG scheduling algorithms designed for single DAG and single machine settings are not well suited for Grid scheduling scenarios, where user run time estimates are available. We show that the proposed new strategies outperform other strategies in terms of approximation factor, mean critical path waiting time, and critical path slowdown. The robustness of these strategies is also discussed.

Keywords Grid computing · Workflow scheduling · Resource management · User run time estimate

A. Hirales-Carbajal · A. Tchernykh (✉)
Computer Science Department,
CICESE Research Center,
Ensenada, BC, México
e-mail: chernykh@cicese.mx

A. Hirales-Carbajal
e-mail: ahirales@cicese.mx

R. Yahyapour · J. L. González-García · T. Röblitz
GWDG – University of Göttingen,
37077 Göttingen, Germany

R. Yahyapour
e-mail: ramin.yahyapour@gwdg.de

J. L. González-García
e-mail: jose-luis.gonzalez-garcia@gwdg.de

T. Röblitz
e-mail: thomas.roeblitz@gwdg.de

J. M. Ramírez-Alcaraz
Colima University, C.P. 28040 Colima, Col., México
e-mail: jmramir@ucol.mx

1 Introduction

The problem of scheduling jobs with precedence constraints is an important problem in scheduling theory and has been shown to be NP-hard [1]. It arises in many industrial and scientific applications. The manner in which a job allocation can be done depends not only on their properties and constraints, but also on the nature of the infrastructure which may be subject to an unpredictable workload generated by independent users in a distributed environment. While

scheduling for Grids has been subject to research for many years, we see a trend to more complex workflows as the Grid has become a common production environment in the scientific field.

Most of the studies have addressed scheduling of a single DAG on a single computer. Only few studies have considered scheduling of multiple DAGs. Similarly, workflow scheduling has also been considered for Grids over the past decade. Several algorithms have been proposed for different types of Grids. Details are discussed in Section 3.

There are two major drawbacks of these approaches: they are based on the premise of knowledge of the exact execution times and consider a single optimization criterion.

In this paper, we consider workflow scheduling with no or estimated runtime information, and multiple optimization goals.

User run time estimates available in many real grid scenarios have been shown to be quite inaccurate [23], and may result in bad workload distribution or inefficient machine utilization [24]. We evaluate whether this information has any practical benefit for workflow scheduling. To our knowledge, this issue has not been addressed in other studies. In addition we propose two strategies which are free of run time information.

Most strategies only consider a single criterion, like total completion time, overall resource utilization, or slowdown. The multi objective workflow allocation problem has rarely been considered so far. Especially the Grid scenario contains aspects which are multi-objective by nature. For instance, system performance related issues and user QoS demands must be considered. Moreover, resource providers and users have different, often conflicting performance goals: from minimizing response time to optimizing resource utilization. Grid resource management should include multiple objectives and use multi-criteria decision support in a way that both stakeholder demands are satisfied. In this paper, we propose multi-criteria analysis of workflow scheduling algorithms as well as the applicability of user job run time estimates in Grid environments.

Another factor that should be considered in Grid scenarios is the unpredictable workload generated by other workflows due to the distributed multi-user environment. Hence, we also propose strategies that take into account both, workflow properties and current site state information.

More precisely, we consider workflow job scheduling in Grids with two levels. At the first level, jobs are allocated to a suitable Grid site, while local scheduling is independently applied to each site on the second level. The first level is a part of the Grid broker and is often called a Grid-layer scheduler. It typically has a general view of job requests while specific details on the state of the resources remain hidden from it.

We address offline scheduling of workflows under consideration of zero release times of workflows and a non-clairvoyant execution, where the scheduler has no knowledge of the real execution length of the tasks in the workflow. While real Grid systems exhibit online behavior, it is known that jobs typically remain in queues for a significant time. Therefore, an offline scheduling strategy is beneficial for the efficient online scheduling of such a set of workflows in the current queue. Moreover, many offline scheduling algorithms exhibit a good performance also in the online case. From theory, it is known that the performance bounds of offline scheduling strategies can be approximated for the online case [25]. As there are yet no established online workload traces of workflows, we start with existing job traces, and model the missing workflows features. We present and evaluate two novel group of workflow scheduling strategies named *MWGS4* (Multiple Workflow Grid Scheduling 4 stages) and *MWGS2* (Multiple Workflow Grid Scheduling 2 stages). The stages of these strategies consist of labeling (*WGS_Label*), adaptive allocation (*WGS_Alloc*), prioritization (*PRIO*), and parallel machine scheduling (*PS*).

We continue this paper by formally presenting our Grid scheduling model in Section 2, and discuss related work in Section 3. We introduce workflow scheduling algorithms and classify them in Section 4. Experimental setup, performance analysis methodology, and experimental results are presented in Section 5. We evaluate scheduling strategies with respect to the robustness of the schedules they produce in Section 6. Finally, we conclude with a summary and an outlook of future works in Section 7.

2 The Scheduling Model

First, we address an offline (deterministic) non-preemptive, non-clairvoyant multiple parallel workflow scheduling problem on a computational grid, where n workflow jobs $J_1, J_2, \ldots, J_n$ must be scheduled on m parallel machines (sites) $N_1, N_2, \ldots, N_m$. Let m_i be the size of machine N_i (number of identical processors), and $m_{1,m}$ be the number of processors in the Grid. Jobs are scheduled on a job-by-job basis, no rescheduling is allowed.

A workflow is a composition of tasks subject to precedence constraints. Workflows are modeled by a directed acyclic graph $G_j = (V_j, E_j)$, where V_j is the set of tasks, and $E_j = \{(T_u, T_v) | T_u, T_v \in V_j, u \neq V\}$, with no cycles $T_u \rightarrow T_v \rightarrow T_u$, is the set of edges between tasks in V_j. Each edge $(T_u, T_v)E_j$ represents the precedence constraint between tasks T_u and T_v, such that T_u must be completed prior execution of T_v is initiated.

Each workflow job J_j is described by the tuple $(G_j, size_j, p_j, p_j^G, \hat{p}_j, \hat{p}_j^G, cpn_j)$: with $G_j = (V_j, E_j)$; its size $size_j$ that is referred to as workflow processor requirements or maximum degree of parallelism; critical path execution time p_j; total workflow execution time p_j^G, user critical path run time estimate $\hat{p}_j$, workflow run time estimate $\hat{p}_j^G$, and number of tasks in the critical path cpn_j. The graph size (width) is the cardinality of the largest set of nodes in G_j that are disjoint (there is no path connecting any pair of them). Such a set represents nodes that might be simultaneously executed.

Each workflow task T_k is a sequential application and is described by the tuple $(r'_k, p'_k, \hat{p}'_k)$: with release date r'_k, execution time p'_k, and user run time estimate $\hat{p}'_k$. Due to the offline scheduling model release date of a workflow $r_j = 0$, however, the release date of a task r'_k is not available before the task is released. Tasks are released over time according to the precedence constraints. A task can start its execution only after all its dependencies have been satisfied. At its release date, a task must be immediately and irrevocably allocated to a single machine. However, we do not demand that the specific processor is immediately assigned to a task at its release date; that is, the processor allocation of a task can be delayed. We use $g(T_k) = N_i$ to denote that task T_k is allocated to machine N_i, and n_i to denote the number of tasks allocated to the site N_i. A machine must execute a task by allocating a processor for an uninterrupted period of time p'_k to it. We use s_k^i to denote start time of the task T_k on machine N_i.

Total workflow processing time p_j^G and critical path execution cost p_j are unknown until the job has completed its execution. They represent time requirements of all tasks of the job $p_j^G = \sum_{T_v \in V_j} p'_v$, and time requirements of tasks that belong to the critical path $p_j = \sum_{T_v \in cp} p'_v$.

We allow multisite workflow execution; hence, tasks of a job J_j can be run on different sites. We consider jobs that tolerate latency since sites may not be at the same geographical location. We also assume that the resources are stable and dedicated to the Grid.

We focus on the analysis of scheduling systems where all jobs are given and processed in the same batch. A set of available and ready jobs will be executed up to the completion of the last one. All jobs that arrive during this time interval will be processed in the next batch. This allows the transformation of the online scheduling problem to be solved in an offline fashion. A relation between this scheme and the scheme where jobs are released over time, either at their release date, or according to the precedence constraints is known, and studied for different scheduling strategies for general or restricted cases [25]. There are several production systems in use whose primary focus is job batching and resource scheduling, like Condor [26], and PBS (Portable Batch System) [27]. They are supported as a job scheduler mechanism by several meta schedulers, like GRAM (Grid Resource Allocation Manager) [28].

The following criteria are used to evaluate proposed algorithms: approximation factor, critical path waiting time, and critical path slowdown. Let c_j and c'_k be the completion time of job J_j and task T_k, respectively. Let $C_{\max}^i$ be the maximum completion time of tasks allocated to machine N_i. The approximation factor of the strategy is defined as $\rho = \frac{C_{\max}}{C_{\max}^*}$, where $C_{\max} = \max_{i=1..m} \{C_{\max}^i\}$ is the makespan of a schedule, and $C_{\max}^*$ is the optimal makespan. Waiting time of a task $tw_k = c'_k - p'_k - r'_k$ is the difference between the completion time of the task, its execution time, and its release date.

Waiting time of a critical path $cpw_j = c_j - p_j$ is the difference between the completion time of the job and the length of its critical path. It takes into account waiting times of all tasks in the critical path. Critical path slowdown $cps_j = 1 + \frac{cpw_j}{p_j}$ is the relative critical path waiting time and evaluates the quality of the critical path execution. A slowdown of one indicates zero waiting times for critical path tasks, while a value greater than one indicates that the critical path completion is increased by increasing waiting time of critical path tasks. The approximation factor is used to qualify the efficiency of the scheduling algorithms. We use the approximation factor over makespan criterion as both metrics differ in constants regardless of the scheduler being used. In addition, to estimate the quality of workflow executions two workflow metrics are used. They are commonly used to express the objectives of different stakeholders of Grid scheduling (end-users, local resource providers, and Grid administrators).

3 Related Work

For single parallel computers, several scheduling algorithms for tasks with arbitrary precedence constraints have been studied in the literature [1–6]. Most of them are designed to schedule only a single Directed Acyclic Graph (DAG) at a time.

Similarly, workflow scheduling has also been considered for Grids over the past decade. Several algorithms have been proposed for different types of Grids. This includes incremental workflow partitioning and full graph scheduling strategies [7], scheduling with communication and processing variations [8], QoS workflow scheduling with deadline and budget constraints [9], scheduling data intensive workflows [10], opportunistic workflow scheduling [11], and level based task clustering of data intensive workflows on computational grids [12].

Another approach to schedule workflows is to decompose them into sub-graphs, which are then treated as moldable tasks [13, 14]. This can be used to schedule multiple workflows, where workflows are compacted into moldable tasks. This method reduces the problem of scheduling multiple workflows to scheduling multiple moldable tasks. Once a moldable task schedule is obtained, the constraints of the workflows can be removed, and tasks can be backfilled to further improve efficiency.

Workflow scheduling has diversified into many research directions: analysis of workflow structure properties in order to identify tasks clusters; minimization of critical path execution time; selection of admissible resources; allocation of suitable resources for data intensive workflows; scheduling subject to QoS constraints, fine tuning workflow execution and performance analysis, etc. [7–12, 15]. Most of them have considered single workflow scheduling problems.

Only few studies have addressed scheduling of multiple DAGs. In [16], authors discussed clustering DAG tasks into chains and allocating them to single machines. They proposed four scheduling heuristics that differ in the way how several DAGs are scheduled. DAGs can either be scheduled independently by interleaving their execution, or can be combined into a single DAG. In [17], a fairness based approach for scheduling of multiple DAGs on heterogeneous multiprocessor is proposed. Two strategies are considered: Fairness Policy based on Finishing Time (FPFT) and Fairness Policy based on Concurrent Time (FPCT). Both strategies arrange DAGs in ascending order of their slowdown value, select independent tasks from the DAG with minimum slowdown, and schedule them using Heterogeneous Earliest Finishing Time (HEFT) [18] or Hybrid.BMCT [19]. FPFT re-calculates the slowdown of a DAG each time a task of the DAG completes execution, while FPCT re-computes the slowdown of all DAGs each time any task in a DAG completes execution. In [20], online scheduling of multiple DAGs is addressed. Authors proposed two strategies based on aggregating DAGs into a single DAG. A modified FCFS and Service-On-Time (SOT) scheduling are applied. FCFS appends arriving DAGs to an exit node of the single DAG, while SOT appends arriving DAGs to a task whose predecessors have not completed execution and will be ready afterward. Once the single DAG has been built, scheduling is carried out by HEFT.

In [21], authors focused on developing strategies that provide a proper distribution of resources among Parallel Task Graphs (PTG), providing fairness and makespan minimization. Constraints are defined according to four general resource sharing policies: unbounded Share (*S*), Equal Share (*ES*), Proportional Share (*PS*), and Weighted Proportional Share (*WPS*). *S* policy uses all available resources. *ES* policy uses equal resources for each PTG. *PS* and *WPS* use resources proportional to the work of each PTG, where the work is considered as critical path cost by width of PTG.

In [22], an Online Workflow Management (OWM) strategy for scheduling multiple mix-parallel workflows is proposed. OWM includes three stages: workflow scheduling, task scheduling, and resource allocation. Workflow scheduling, referred to as Critical Path Workflow Scheduling (CPWS), labels DAG tasks, sorts them, and stores into independent buffers. Labeling is based on the upward rank strategy. The sorting arranges tasks in descendent order of the task rank. Task scheduling referred to as a rank hybrid phase determines the task execution order. Tasks are sorted in descending order when all tasks in the queue belong to the same workflow. Otherwise, they are sorted in ascending order. Allocation assigns idle processors to tasks from the waiting queue. Tasks with highest priority are allocated only if their resource requirements can be satisfied subject to minimize earliest estimated finishing time.

4 Proposed Workflow Scheduling Strategies

In this section, we present details of our scheduling strategies. We consider multi-stage scheduling named *MWGS4* (Multiple Workflow Grid Scheduling with 4 stages) and *MWGS2* (Multiple Workflow Grid Scheduling with 2 stages). The stages of these strategies consist of (1) labeling (*WGS_Label*), (2) adaptive allocation (*WGS_Alloc*), (3) prioritization (*PRIO*), and (4) parallel machine scheduling (*PS*). Hence, we regard them as $MWGS4 = WGS\_Label + WGS\_Alloc + PRIO + PS$ and $MWGS2 = WGS\_Alloc + PS$.

- At the *WGS_Label* stage, workflow tasks are labeled by their ranks using user run time estimates.
- At the *WGS_Alloc* stage, the list of tasks that are ready to be started is maintained. Tasks from the list are allocated to suitable resources using a given optimization criteria.
- At the *PRIO* stage, labels are used to prioritize tasks.
- At the *PS* stage, a *PS* algorithm is applied to tasks that were allocated during *WGS_Alloc* stage for each parallel machine independently.

Note that *WGS_Label* procedures are designed to schedule only a single workflow at a time. To study an impact of task labeling and task allocation policies in the multiple workflow scheduling environment with unpredictable workload, we compare *MWGS4* with two stage strategies $MWGS2 = WGS\_Alloc + PS$, where tasks labeling is not performed and prioritizing is not used. Hence, we compare *MWGS4* with workflow scheduling strategies that are based only on allocation policies designed for scheduling independent tasks.

4.1 Task Labeling (WGS_Label)

Tasks labeling (*WGS_Label*) prioritizes workflow tasks considering each workflow independently. Labels are not changed nor recomputed on completion of predecessor tasks. Task labels are used to identify properties of a given workflow, such as the task level, critical path length, etc. We distinguish two labeling strategies: Downward Rank (*DR*), and Constant Rank (*CR*). *DR* estimates the length of the longest path from considered task to a terminal task in a workflow. Descending order of *DR* supports scheduling tasks on the critical path first.

CR labels tasks using the following procedure: all tasks of the critical path are labeled by the critical path cost. Then the critical path tasks are removed together with corresponding edges. The procedure is repeated recursively for each graph of a residual forest.

Figure 1 illustrates the *CR* labeling procedure. In Fig. 1a critical path nodes $\{A, C, F, I\}$ (dash lines), with total length 11 are labeled by 11 and removed. A dummy zero cost task $\{P\}$ is added to the residual graph, and nodes with no predecessors are used as successors (Fig. 1b). New critical path nodes $\{P, E, G\}$, with length 7 are labeled by 7 and removed. The resulting DAG is shown in Fig. 1c. The process is repeated until the residual graph vertex set V_j is empty. User run time estimates are used for labeling. Descending order of *CR* supports scheduling all tasks on the longer path first.

4.2 Task Allocation (WGS_Alloc)

The list of tasks that are ready to be started is maintained. Independent tasks with no predecessors and with predecessors that completed their execution are entered into the list. Upon completion of a workflow task its immediate successors may become available and are entered into the list. Allocation policies are responsible for selecting a suitable site for task allocation. We use adaptive task allocation strategies presented in [24, 29] as independent task allocation policies, and introduce novel strategy MaxAR (Table 1). We distinguish allocation strategies depending on the amount of information they require. Four levels of available information are considered.

Level 1: Information about the status of available resources is available.
Level 2: Once a task has been submitted, the site with least load per processor is known.
Level 3: All information of the level 2 and the job run time estimates are available.
Level 4: Information of the level 3, all local schedules, and site status are available. Levels 1–4 information may be provided via Grid information service. Note that the number and the sizes of the parallel machines are known.

In addition, the following single workflow scheduling strategies, used in many performance evaluation studies, are considered: *HEFT* (Heterogeneous Earliest Finishing Time First), and *CPOP* (Critical Path on Processor) [18].

HEFT schedules DAGs in two phases: job labeling and processor selection. In the job labeling phase, a rank value (upward rank) based on mean computation and communication costs is assigned to each task of a DAG. In the following, we consider only computation costs.

CPOP labeling phase computes the rank of each task as the sum of its up and downward ranks. It identifies tasks on the critical path as those having equal cost.

Prior the job allocation phase, *CPOP* selects a processor that minimizes its computation cost. Such a processor is referred to as the critical path

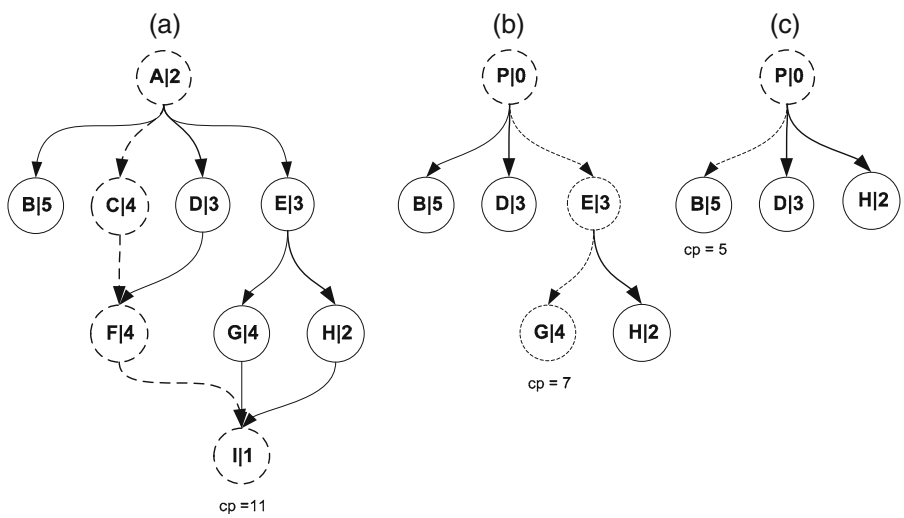

Fig. 1 CR labeling of tasks

Table 1 Task allocation strategies

MPS_Alloc	Level	Description
MaxAR	1	Allocates task T_k to the site with maximum fraction of available resources at time r'_k: $\max_{i=1...m}\left(\frac{avail_i}{m_i}\right)$, were $avail_i$ denotes the number of available processors at time r'_k. MaxAR does not consider the requirements of queued jobs
Rand	1	Allocates task T_k to a machine with the number randomly generated from a uniform distribution in range $[1, m]$
MLp	2	Allocates task T_k to the site with least load per processor at time r'_k $\min_{i=1...m}(n_i/m_i)$
MLB	3	Allocates task T_k to the site with least work per processor at time r'_k: $\min_{i=1...m}\left(\sum_{g(T_k)=N_i}\frac{p'_k}{m_i}\right)$
MWT	4	Allocates task T_k to the site with minimum average task waiting time $\min_{i=1...m}\left(\sum_{g(T_k)=N_i}\frac{tw_k}{n_i}\right)$
MST	4	Allocates task T_k to the site with earliest start time for this task $\min_{i=1...m}\left(s^i_k\right)$
MCT	4	Allocates task T_k to the site with earliest site completion time $\min_{i=1...m}\left(c^i_{\max}\right)$ before allocating task T_k

processor. On the allocation phase tasks that are on the critical path are assigned to the critical path processor, otherwise, they are allocated to a processor that minimizes the task earliest finishing time. Since knowledge of local schedules is required by both *HEFT* and *CPOP* their allocation strategies are categorized as level 4.

4.3 Local Queue Prioritization (PRIO) and Site Scheduling Algorithm (PS)

Local Resource Management System (*LRMS*) prioritizes tasks in the waiting queue based on their labels and assigns tasks with the highest priority to execution. To this end, *LRMS* applies local queue ordering *PRIO* procedure. Two local queue prioritizing policies are distinguished: increasing and decreasing, referred as *Shortest Chain First* (*SCF*), and *Longest Chain First* (*LCF*), hence, *PRIO* {*SCF, LCF*}. A decreasing order allows high priority tasks to be schedule first, while an increasing ordering schedules low priority tasks first. For tasks belonging to a critical path, an increasing order schedules longest critical path tasks first, while a decreasing order schedules shortest critical path tasks first. The labeling stage is omitted when workflow properties are not used for job prioritization.

Different scheduling algorithms may be used by the *LRMS*. We assume that the *LRMS* uses the on-line parallel scheduling algorithm *EASY* (First-Come-First-Serve with *EASY* backfilling). In order to apply *EASY* backfilling user run time estimates are used.

5 Experimental Validation

In order to provide performance comparison, we use workloads from a parametric workload generator that produces workflows that resemble those of real workflow applications such as Cybershake, Epigenomics, Gnome, Inspiral, Ligo, Montage, and Sipht [30]. We show that known single DAG scheduling algorithms are not suitable for Grid multiple workflow scheduling contexts, while our *MWGS4* approach delivers good performance and outperforms other algorithms in terms of approximation factor, mean critical path waiting time, and critical path slowdown. However, one of their drawbacks is that these strategies are based on labeling and ordering of site local queues, and have significant computational complexity.

We show that quite simple schedulers *MaxAR* with minimal information requirements can provide good performance for multiple workflow scheduling with user run time estimates. Note, that it does not use specific workflow information nor requires task labeling. Besides the performance aspect it does not require additional management overhead such as DAG analysis,

requesting info about local schedules, and constructing preliminary schedules by the Grid broker.

While machines in real Grids often exhibit different forms of heterogeneity, like different hardware, operating system, software, dynamic behavior, many different types of jobs, and other restrictions, the model of any study on Grid scheduling must be an abstraction of reality to perform performance evaluation in a repeatable and controllable manner. On the other hand, key properties of Grids should be observed to provide benefits for real installations. As computational Grids are successors of parallel computers, we extend their most basic model of Garey and Graham [31] that assumes identical processors as well as jobs with unknown processing times. Hence, we restrict ourselves to heterogeneous machines with a different number of the identical processors or cores, as architectures of individual cores and their clock frequency tend to be rather similar. This simplified model neither matches every real installation nor all real applications, but the assumptions are nonetheless reasonable. The proposed algorithms may serve as a starting point for algorithms that can be implemented in real computing Grids.

5.1 Experimental Setup

Several Grid scheduling toolkits with different functionality have been developed in recent years. Most of them are used by their own developers, target a specific community, and designed to study specific features, such as data replication, P2P applications, etc. [43]. GridSim [41] and SimGrid [42] are freely available, widely used and acknowledged frameworks for the simulation of distributed resource management, and scheduling. GridSim was initially intended for grid economy scheduling, and later on became used in other areas of grid computing. Likewise, the SimGrid scope has been generalized from community-specific toolkit. However, validating the performance of multiple workflow scheduling strategies needs flexibility to manage Grid scenarios with unpredictable workloads generated by other workflows due to the distributed multi-user environment. To this end, we require a grid simulation tool that is able to take into account both, workflow properties and current site state information.

All experiments are performed using the Grid scheduling simulator *tGSF* (Teikoku Grid Scheduling Framework). *tGSF* is a standard trace based simulator that is used to study Grid resource management problems. We have extended Teikoku to include Grid workflow job scheduling capabilities. Design details of the simulator are described in [32]. Two workloads types are used in experiments for comprehensive analysis: workload A (306 workflows) and workload B (508 workflows). Their properties are described in Appendix A.1.

Note that the number of tasks in workload B (120300 tasks) is almost five times as much as in workload A (25002 tasks). The resource consumption is also five times as much. It gives us two different scenarios for simulation. The Cybershake, Epigenomics, Genome, Inspiral, LIGO, Montage, and SIPHT workflows [33] are used. They are publicly available via the Pegasus project portal [30], and include $|V_j|$, $|E_j|$, $size_j$, p_j, p_j^G and cpn_j. Examples of workflow types are shown in Appendix A.1 (Fig. 4). Their characteristics are presented in Appendix A.1 (Tables 7 and 8). These workflows were executed using the Pegasus Workflow Management System (Pegasus-WMS) on the TeraGrid.

In this paper, we also use TeraGrid sites information for an experimental analysis. We chose the TeraGrid since it has been used in workflow scheduling studies [14]. Existing TeraGrid workloads are recognized to be intensive, complex and multidimensional in terms of job type, job size and job duration. To make possible a variety of user and usage scenarios with multiple workflow workloads, we have chosen only three computational sites with a total of 5376 processors (Table 2), which compromise among the three smallest machines in the TeraGrid. Background workload (locally generated jobs) that is an important issue in non-dedicated Grid environment is not addressed. Communication cost is not considered.

Users of large parallel computers are typically required to provide run time estimates for submitted jobs. It is used by production system schedulers to bind job execution time, and to avoid

Table 2 Experimental testbed

Description	Settings
Workload type	Workflows. Workload type A and B (see Appendix A.1).
Number of grid sites	3
Site sizes	Sizes of sites are taken from the TeraGrid: site 1 (Bigred), 3072 procs; site 2 (Ember), 1536 procs; and site 3 (Pople), 768 procs; 5376 procs in total
Metrics	Mean critical path waiting time, critical path slowdown, and approximation factor
Number of experiments	30
Experimental unit	LiDO Linux-HPC-Cluster at the Technische Universität Dortmund
Problem model	Offline

wasting resource consumption. Thus, when a job reaches such a limit, it is typically aborted. To our knowledge, run time estimates for workflows are not provided. Thus, for the simulation purposes, we set $\hat{p}_j$ to $p_j + c * p_j$, where c is randomly generated value uniformly distributed between 0 and 5, and $\hat{p}'_k$ to $\hat{p}_j$, for each $T_k \in V_j$, modeling inaccuracy of the user run time estimates as proposed in [34]. Note that only level 3 and 4 allocation strategies use run time estimates.

Table 3 summarizes site scheduling strategies, site queuing policies, workflow scheduling strategies, and workflow labeling policies. Two class of strategies *MWGS2* and *MWGS4* are evaluated.

5.2 Performance Analysis

A good scheduling algorithm should schedule jobs to achieve high Grid performance while satisfying various user demands in an equitable fashion. Often, resource providers and users have different, conflicting, performance goals: from minimizing response time to optimizing resource utilization. Grid resource management involves multiple objectives and may use multi-criteria decision support, for instance, based on the Pareto optimality. However, it is very difficult to achieve fast solutions needed for Grid resource management by using the Pareto dominance.

The problem is very often simplified to a single objective problem or to different methods of objectives combining. There are various ways to model preferences, for instance, they can be given explicitly by stakeholders to specify an importance of every criterion or a relative importance between criteria. Due to the different nature of criteria, the actual difference may have a different meaning.

In order to provide effective guidance in choosing the best strategy, we performed a joint analysis of several metrics according to methodology used in Ramírez-Alcaraz et al. [24].

They use an approach to multi-criteria analysis assuming equal importance of each metric. The goal is to find a robust and well performing strategy under all test cases, with the expectation that it will also perform well under other

Table 3 Experimental settings

	MWGS model	Parameters	Description
I	WGS_Alloc+PS	Labeling is not performed *PRIO* = FCFS PS = EASY	9 scheduling strategies are analyzed: MLp, MaxAR, MLB, MWT, MCT, MST, CPOP, Rand, and HEFT
II	WGS_Label+ MPS_Alloc+ RIO+PS	WGS_Label∈ {DR, CR}, *PRIO*∈{SCF, LCF}, PS = EASY	16 workflow scheduling strategies and 3 best rigid job strategies are analyzed: DR+MLp+LCF, DR+MaxAR+LCF, DR+MLB+LCF, DR+MST+LCF DR+MLp+SCF, DR+MaxAR+SCF, DR+MLB+SCF, DR+MST+SCF CR+MLp+LCF, CR+MaxAR+LCF, CR+MLB+LCF, CR+MST+LCF CR+MLp+SCF, CR+MaxAR+SCF, CR+MLB+SCF, CR+MST+SCF MLp, MaxAR, MLB

conditions, e.g., with different Grid configurations and workloads.

The analysis is conducted as follows. First, we evaluate the degradation in performance of each strategy under each of the three metrics (Table 4).

They are well known performance metrics commonly used to express the objectives of different stakeholders of Grid scheduling (end-users, local resource providers, and Grid administrators). The objective of the mean critical path waiting time and slowdown performance metrics is to evaluate the quality of the critical path execution in the context of scheduling several workflows.

Note that in the experimental analysis the lower bound of the optimal completion time $\breve{C}^*_{\max} = \max\left\{\max_j\left(p_j\right), \frac{\sum_{j=1,n}\left(p_j^G\right)}{m_{1,m}}\right\}$ is used instead of the optimal makespan $C^*_{\max}$ to calculate the approximation factor. The degradation in performance is computed relatively to the best performing strategy for the metric, as follows: $\left(\frac{strategy\ metric}{best\ metric} - 1\right) \cdot 100$.

Thus, each strategy is now characterized by 3 numbers, reflecting its relative performance degradation under the test cases. In [24], these 3 values (assuming equal importance of each metric) are averaged and ranked. The best strategy, with the lowest average performance degradation, has rank 1. However, some metrics might have little variation by nature or by a given scenario, and the averaging would yield to lesser impact to the overall score. In this paper, a different approach is taken. We average the individual ranks instead of the metric degradations their self.

Note that we try to identify strategies which perform reliably well in different scenarios; that is we try to find a compromise that considers all of our test cases. For example, the rank of the strategy in the average performance degradation could not be the same for any of the metrics individually.

Table 4 Metrics

Metric	Definition
Approximation factor	$\rho = \frac{C_{\max}}{\breve{C}^*_{\max}}$
Mean critical path waiting time	$\overline{cpw} = \frac{1}{n}\sum_{j=1}^{n} cpw_j$
Mean critical path slowdown	$\overline{cps} = \frac{1}{n}\sum_{j=1}^{n} cps_j$

5.3 Experimental Results

In this section, 2- and 4-stage scheduling systems are simulated by various types of input processes and constraints. The following simulation results compare the performance of the 25 strategies (6 *MWGS2* strategies designed for independent tasks, 2 workflow scheduling strategies, Rand, and 16 *MWGS4* priority based workflow scheduling strategies). We conduct their comprehensive performance evaluation for two different scenarios considering three metrics (Tables 3 and 4). In the experiment I, analysis of 9 two-stage strategies is performed based on the workload *A*. In experiment II, we select the 3 best performed strategies of the previous experiment, add 16 four-stage workflow strategies and analyze them with the heavier load *B*.

5.3.1 Scheduling Strategies

In this section, we evaluate scheduling strategies *MWGS2* = *WGS_Alloc+PS* using workload *A* (Table 3). Firstly, we compare allocation strategies considering the approximation factor, mean critical path waiting time, and mean critical path slowdown, independently. Then, we perform a joint analysis of these metrics according to methodology described in Section 5, and present their ranking when considering all metrics average.

Table 5 shows the performance degradation of all strategies for ρ, $\overline{cpw}$, and $\overline{cps}$. It also shows the mean degradations of the strategies when considering all metrics average, and ranking for all test cases. A small percentage of degradation indicates that the performance of a strategy for a given metric is close to the performance of the best performing strategy for the same metric. Therefore, small degradations represent better results.

From Table 5 we observe that *MCT* and *MWT* are strategies with worst approximation factors with 4717 and 3005% of degradation in performance. One of the reasons for such large values

Table 5 Rounded performance degradation

Metric	Strategy								
	CPOP	HEFT	MCT	MLB	MLp	MST	MWT	MaxAR	Rand
ρ	8	5	4717	0	0	5	3005	0	0
$\overline{cps}$	61	50	98	5	0	50	98	3	10
$\overline{cpw}$	15470095	9823658	789285050	56269	9	9862349	1167811792	13862	143334
Mean	5156721	3274571	263096622	18758	3	3287468	389271632	4622	47781
Ranking	7	5	8	3	1	6	9	2	4

is high inaccuracy of user run time estimates used in scheduling. Such a problem is also addressed in [24, 34]. *MLp* strategy has small percent of degradations in all metrics. We see that ρ and $\overline{cps}$ metrics have less variation compared with $\overline{cpw}$, and yield to lesser impact to the overall score. The approximation factor of *MLB*, *MLp*, *MaxAR*, and *Rand* are near the lower value. *MLp*, *MaxAR* and *MLB* have similar behavior in terms of critical path slowdown. Due to our model is a simplified representation of a system, we can conclude that these strategies might have similar efficiency in real Grid environment when considering above metrics.

However, there exist significant differences comparing $\overline{cpw}$, where *MLp* shows the best performance. In this strategy the critical path completion time does not grow significantly with respect to p_j, therefore tasks in the critical path experience small waiting times. Results also show that for all strategies small mean critical path waiting time degradation corresponds to small mean critical path slowdown.

We presented the metrics degradations averages to evaluate performance of the strategies, and showed that some strategies tend to dominate results. The degradation approach provides the percent improvement, but does not show negative effects of allowing a small portion of the problems to dominate the conclusions.

To analyze possible negative effects of allowing a small portion of the problem instances with large deviation to dominate the conclusions that based on averages, and to help with the interpretation of the data generated by the benchmarking process, we presented performance profiles of our strategies in Fig. 2.

The performance profile $\rho(\tau)$ is a non-decreasing, piecewise constant function that presents the probability that a performance ratio $r = \frac{strategy\ metric}{best\ metric}$ is within a factor τ of the best ratio [35, 36].

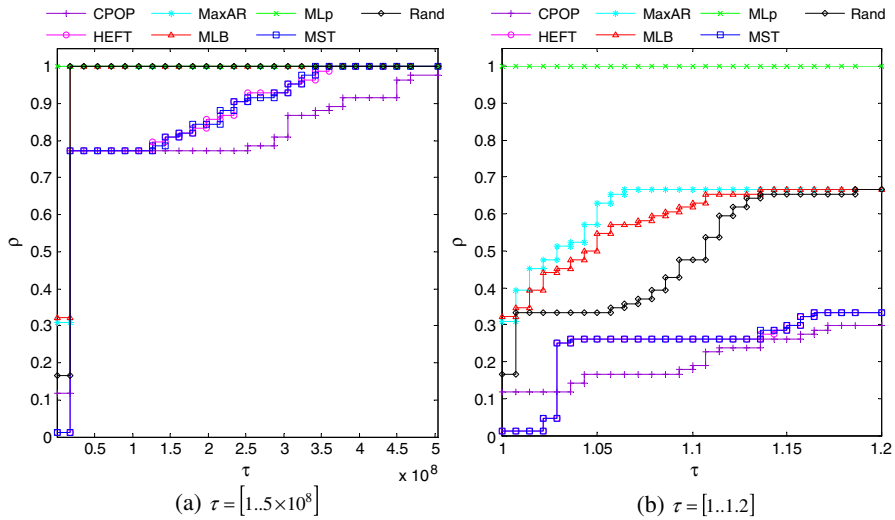

Fig. 2 Performance profiles

The function $\rho(\tau)$ is the cumulative distribution function. Strategies with large probability $\rho(\tau)$ for smaller τ are to be preferred. For instance, in Fig. 2b, $\rho(1.08) = 0.65$ means that *MaxAR* performed at most 8% worse than the best strategy on 65% of the instances considered.

Figure 2 shows the average performance profile $\overline{\rho}(\tau) = \frac{1}{3}\sum_{s=1\ldots3}\rho(\tau)$ of three metrics assuming equal importance of each metric, where $\tau \in [1..r'_M]$, and r'_M is the maximum over all r_M values ratio [35]. We show the performance profiles in different ranges of τ to provide objective information for analysis of a test set. Figure 2a and b show the performance profiles of the 7 strategies in the interval $\tau = [1..5 \times 10^8]$ and $\tau = [1..1.2]$, respectively. This figure displays the large discrepancies in the performance ratios on a substantial percentage of the problems.

MLp has the highest ranking and the highest probability of being the better strategy. The probability that it is the winner on a given problem within factors of 1.0–1.2 of the best solution is about 1. If we choose being within a factor of 1.07 as the scope of our interest, then the probability that the low complexity strategy *MaxAR* is the winner is 0.65. If we choose being within a factor of 1.15 of the best solution then either *MaxAR, MLB* or *Rand* would suffice with a probability 0.65.

The most significant aspect of Fig. 2 is that on this test set *MLp* dominates other strategies: its performance profile lays above all others for all values of performance ratios. We see that an appropriate distribution of jobs over the Grid has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules, even when specialized workflow scheduling strategies are used. It turns out that *MLp* is the best performing algorithm for several workflow scheduling. *MaxAR* provides second best result. They take into account dynamic site state information. Note, that they do not use specific workflow information nor require task labeling. Besides the performance aspect the use of these strategies does not require additional management overhead such as DAG analysis, and info about local schedules.

5.3.2 Priority Based Workflow Scheduling Strategies

In the experiment II, we employ workflow properties for scheduling purposes. We evaluate 16 priority based workflow scheduling strategies $MWGS4 = WGS\_Label + WGS\_Alloc + PRIO + PS$, where two DAG labeling algorithms *CR* and *DR* (Section 3) are used, and compare them with 3 non-priority based scheduling strategies $MWGS2 = WGS\_Alloc + PS$ for comprehensive analysis. *WGS_Label* procedure generates labels that are used as tasks priorities. *PRIO* procedure uses these priorities to sort site local queues. Results are presented in Table 6.

We find that *MLp* and *MaxAR* with different labeling and prioritization policies are more beneficial than other strategies. Their average degradations are less than 18.6%.

Results show that labeling strategies *DR* and *CR* affect very slightly the performance of the workflow scheduling strategies in a Grid setting. We found that classical single DAG scheduling algorithms *CPOP* and *HEFT* designed for single machine settings are not suitable for Grid scheduling contexts. Workflow scheduling algorithms that use task allocation strategies based on user run time estimates give unstable results.

As expected, *LCF* prioritization performs well for minimization of approximation factor in *MLB*, *MLp*, and *MaxAR* allocation strategies indistinctive of the labeling strategies used. While *SCF* decreases mean critical path slowdown, and mean critical path waiting time.

Results show that *CR* and *DR* labeling reduce degradation in most cases. *DR* labeling performs better than *CR* for each metric and all metrics average. *SCF* ordering performs better than *LCF*, for considered metrics.

To analyze possible biasing the results and avoid negative effects of allowing a small portion of the problem instances with large deviation to dominate the conclusions, we presented performance profiles $\overline{\rho}(\tau)$ of our strategies in Fig. 3.

To reduce complexity of presented results, we show only 7 strategies with the ranking from 1 to 4: (2) $dr + maxar + lcf$, (5) $dr + mlp + scf$, (10)

Table 6 Performance degradations of all strategies

	Strategy	ρ	$\overline{cps}$	$\overline{cpw}$	Mean	Ranking
2	DR+MaxAR+LCF	0	0	0	0	1
5	DR+MLp+SCF	0	0	0	0	1
10	CR+MaxAR+LCF	0	0	0	0	1
13	CR+MLp+SCF	0	0	0	0	1
18	MaxAR	7.9	0	0	2.6	2
9	CR+MLp+LCF	0	3.8	24.5	9.4	3
14	CR+MaxAR+SCF	2.1	4.6	30	12.2	4
1	DR+MLp+LCF	0	4	48.2	17.4	5
6	DR+MaxAR+SCF	1.8	7	47.1	18.6	6
17	MLp	13.9	5.3	51.9	23.7	7
7	DR+MLB+SCF	11.5	1.7	71.2	28.1	8
15	CR+MLB+SCF	11.8	4.6	74.6	30.3	9
11	CR+MLB+LCF	0	9.5	190.6	66.7	10
3	DR+MLB+LCF	0	10	234.9	81.6	11
19	MLB	18.2	14.3	219.7	84.1	12
16	CR+MST+SCF	51.1	22.7	558.1	210.6	13
8	DR+MST+SCF	50.7	24.1	592.1	222.3	14
12	CR+MST+LCF	513.4	7.3	345.8	288.8	15

cr + *maxar* + *lcf*, (13) *cr* + *mlp* + *scf*, (18) *maxar*, (9) *cr* + *mlp* + *lcf*, (14) *cr* + *maxar* + *scf* (see, Table 6). Figure 3 shows the performance profiles in the intervals $\tau = [1, 10]$ and $\tau = [1, 1.2]$. We see that (5) *DR* + *MLp* + *SCF* has the highest probability of being the better strategy. The probability that it is the winner on a given problem within a factor of 1.1 of the best solution is about 0.70, and within a factor of 1.2 is about 1 (Fig. 3b). It dominates other strategies: its performance profile lays above all others. It also has ranking 1 for all metrics degradation average.

If we choose being within a factor of 1.2 as the scope of our interest, then either (5) *DR* + *MLp* + *SCF* would suffice with a probability 0.99, (13) *CR* + *MLp* + *SCF* with a probability 0.7, (14) *cr* + *maxar* + *scf* with probility 0.62, or (18) *MaxAR* with a probability 0.5 (Fig. 3b).

We see that an appropriate distribution of load per processor by *MLp* strategy with priority based

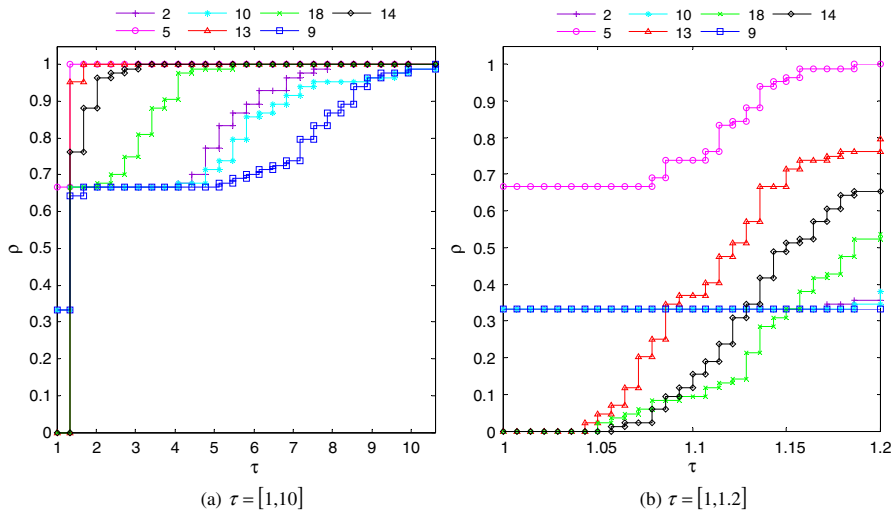

Fig. 3 Performance profiles

(a) $\tau = [1,10]$ (b) $\tau = [1,1.2]$

scheduling has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules, even when specialized workflow scheduling strategies are used.

Strategies (2) $DR + MaxAR + LCF$, (5) $DR + MLp + SCF$, (10) $CR + MaxAR + LCF$, and (13) $CR + MLp + SCF$ have the best ranking considering average degradation of three metrics. Considering the performance profiles, it turns out that $DR + MLp + SCF$ is the best performing algorithms for several workflow scheduling.

Note, that they use specific workflow information, require task labeling, and ordering. Hence, they always suffer from a large time complexity. Moreover, ordering by *PRIO* is responsibility of Grid site policy, and Grid resource management might not have control over it, and local clusters might not perform task ordering.

The degradation of *MaxAR* has a rank of 2, however, its average degradation is only within 2.6% of the best strategy. If we choose being within a factor of 1.2 of the best solution *MaxAR* would suffice with a probability 0.5.

We conclude that in real Grid scenarios this strategy might have similar performance comparing with the best ones. Besides the performance aspect the use of *MaxAR* does not requires additional management overhead such as DAG analysis, and site local queue ordering. It has small time complexity.

6 Robustness

Most of the work in literature has focused on DAG scheduling heuristics that minimize the makespan. Only few studies have evaluated DAG scheduling with respect to the robustness of schedules they produce.

In [37], a bi-objective algorithm for scheduling of independent tasks and DAGs subject to optimizing makespan and reliability is addressed. The RHEFT (Reliable HEFT) strategy is proposed, where tasks are allocated to resources with the smallest value of failure rate and execution time. Authors showed that optimizing both criteria is not successful. Therefore, a user has to find a compromise between minimization of the makespan and maximization of the reliability. In [38], authors extend results of [37] by developing scheduling strategy that increases reliability by replicating tasks. Two fundamental problems are addressed: determining the number of replicas and defining a criterion for a proper allocation of the replicated tasks. Scheduling is performed by a list scheduling based heuristic called Scheduling by Replication (LSR). LSR allocates replicas on the least loaded processors.

In [39], the robustness of twenty DAG scheduling heuristics from literature designed to minimize the makespan was studied. A stochastic model was used to evaluate the standard deviation of the makespan, so that, the higher the deviation, the less robustness of the strategy. Authors showed that static scheduling strategies tend to be more robust. They demonstrated that the makespan and robustness are correlated, and addressed problems of uncertainty due to resource failures.

In [40], authors study an impact of information quality on performance estimation accuracy in different scheduling strategies in terms of performance estimates, by adding a percentage of random noise.

We study another aspect of the robustness of DAG scheduling heuristics in the Grid environment. *MWGS* strategies work under the assumption that when a broker requests site state information, this information is provided instantly and is most recent. In a real Grid scenario, this assumption often does not hold due to several factors such as communication delays, congestion, processing load, etc. An information service is used to collect information from multiple scattered resources. It collects configuration, state, load, etc., and reports them periodically or by request. It is also responsible for assembling this information, handling conflict resolutions, identification of critical events, generating reports, and performing various automated actions.

Once the data have been retrieved, they can be used for resource allocation decisions. If the data are not updated frequently they can become stale. Such a scenario is highly expected when the system load is high. In this section, we study an impact of data retrieving delay on the performance of allocation policies, and on the overall Grid performance. We assume that site state

information on the Broker level is updated at a given time interval. Resource allocation decisions made within this time interval are based on the same potentially out of date information.

Experiments were conducted using workload *B* and strategies which show good performance (see, Section 5.3.1) *MWGS2*, with *WGS_Alloc* {*MLB, MLp*, and *MaxAR*}, and $PS = EASY$.

In order to evaluate their robustness, we incorporate a *site refresh interval* into the broker information service. We use refresh intervals of 0, 2, 4, …, 2^{21} milliseconds (up to 35 min).

The degradation in performance of each strategy under each of the three metrics versus refresh interval is shown in Figs. 9, 10 and 11 in Appendix. Mean performance degradation of each strategy under each of the three metrics versus refresh interval is shown in Fig. 12 in Appendix. We see that workflow allocation strategies under the short delay scenario are invariant to stale information during 0 to 4 s refresh interval.

MaxAR, and *MLp* strategies are more robust for this refresh interval. Performance of strategies deteriorated when refresh interval exceeds 4 s. The degradations vary greatly if refresh interval is in the order of minutes. We conclude that *MaxAR* is robust and well performed strategy.

7 Conclusions

Effective workflow management requires an efficient allocation of tasks to limited resources over time and currently the subject of many research projects. Multiple workflows add a new complexity to the problem. The manner in which allocation of a workflow task can be done depends not only on the workflow properties and constraints, but also on an unpredictable workload generated by other workflows in the distributed context.

In this paper, we concentrate on allocation strategies that take into account both dynamic site state information and workflow properties. We characterize the type and amount of information they require, and analyze strategies that consist of two and four stages: labeling, adaptive allocation, prioritization and parallel machine scheduling.

We conduct a comprehensive performance evaluation study of 25 workflow scheduling strategies in Grids using simulation. In order to provide effective guidance in choosing the best strategy, we performed a joint analysis of three metrics (approximation factor, mean critical path waiting time, and critical path slowdown) according to a degradation methodology that considers multi-criteria analysis assuming equal importance of each metric.

To analyze possible biasing results and negative effects of allowing a small portion of the problem instances with large deviation to dominate the conclusions, we presented performance profiles of our strategies. The goal is to find a robust and well performing strategies under all test cases, with the expectation that it will also perform well under other conditions, e.g., with different Grid configurations and workloads.

Our study results in several contributions.

- Firstly, we identify several levels of information available to make scheduling decisions with respect to workflows and workflow tasks allocations.
- We discussed and analyzed a variety of two- and four-stage allocation strategies with user run time estimates together with EASY backfilling local scheduling algorithm.
- We propose to use adaptive allocation strategies that demonstrate high efficiency as independent job allocation policies and take into account dynamic site state information.
- We demonstrate that DAG scheduling algorithms designed for a single DAG and single machine settings are not suitable for Grid scheduling contexts.
- We show that the information about local schedules traditionally used for workflow allocation like Earliest Finishing Time does not help to improve the outcome of the multiple workflow scheduling strategies when limited information about tasks run time is available.
- Algorithms that use task allocation strategies based on user run time estimates give unstable results.

When we examine the overall Grid performance based on real data, we find that an appropriate distribution of load per processor by *MLp* with

priority based scheduling has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules, even when specialized workflow scheduling strategies are used. However, they use specific workflow information, require task labeling, and ordering. Hence, they always suffer from a significant time complexity. Moreover, ordering by *PRIO* is responsibility of Grid site policy, and Grid resource management might not have control over it, and local clusters might not perform task ordering.

MaxAR is the second best strategy. Its degradation in performance has ranking 2. However, it is only within 2.6% of the best strategy. Even we choose being within a factor of 1.2 of the best solution *MaxAR* would suffice with a probability 0.5.

In real Grid environments this strategy might have similar performance comparing with the best ones when considering above metrics. Besides the performance aspect the use of *MaxAR* does not require additional management overhead such as DAG analysis, site local queue ordering, and constructing preliminary schedules by the Grid broker. It has small time complexity.

As a conclusion, we can draw that for practical purposes quite simple scheduler *MaxAR* with minimal information requirements can provide good performance for multiple workflow scheduling.

Our results consider offline scheduling which can be used as a starting point for addressing the online case. Online Grid workflow management brings new challenges to above problem, as it requires more flexible load balancing workflows and their tasks over the time. Many offline scheduling strategies perform reasonable well when used in online scenarios. The proposed strategies can be easily adapted to solve online multiple workflows scheduling problem without extra computational complexity. However, further study is required to assess their actual efficiency and effectiveness. This will be subject of future work requiring a better understanding of online workflows and, ideally, the availability of Grid logs with workflows from real installations. Currently, such data are not available. Moreover, scheduling in non-dedicated Grid environment, where background workload (locally generated jobs) is unpredictable, is another important issue to be addressed. The communication latency that is a major factor in data Grid scheduling performance is additional relevant issues to be considered.

Acknowledgements Part of this work was supported by UABC under grant #4006/C/35/14, Deutscher Akademischer Austausch Dienst (DAAD) under grants A/07/74928, A/09/03177, and CONACYT under grant #32989-A.

Appendix

A.1 Workload Characteristics

To facilitate evaluation of proposed multiple workflow scheduling algorithms Fig. 4 shows examples of workflow types available in the Pegasus project [30]. Tables 7 and 8 list the characteristics of the used workloads, where n is the number of jobs; $|V_j|$ is the number of tasks in the job; $\overline{cpn_j}$ is the average number of tasks in critical paths; p_j is the critical path cost, and $\overline{size_j}$ is the average width of a given DAG.

Figures 5 and 6 show the critical path run time in the workload A. 83% of workflows have short critical path run time with the range between 28 and 83 min, while the remaining 17% of workflows have a critical path run time within the range of 137–520 min. The width of workflows is within the range of 5 to 162 tasks, and the number of tasks in the job is within the range of 24 to 200.

Figures 7 and 8 show the critical path run time in the workload B. 65% of workflows have short critical path run time with the range between 28 and 83 min, while the remaining 35% of workflows have a critical path run time of 583 min in average. The width of workflows is within the range of 5–578 tasks, and the number of tasks in the job is within the range of 24–700.

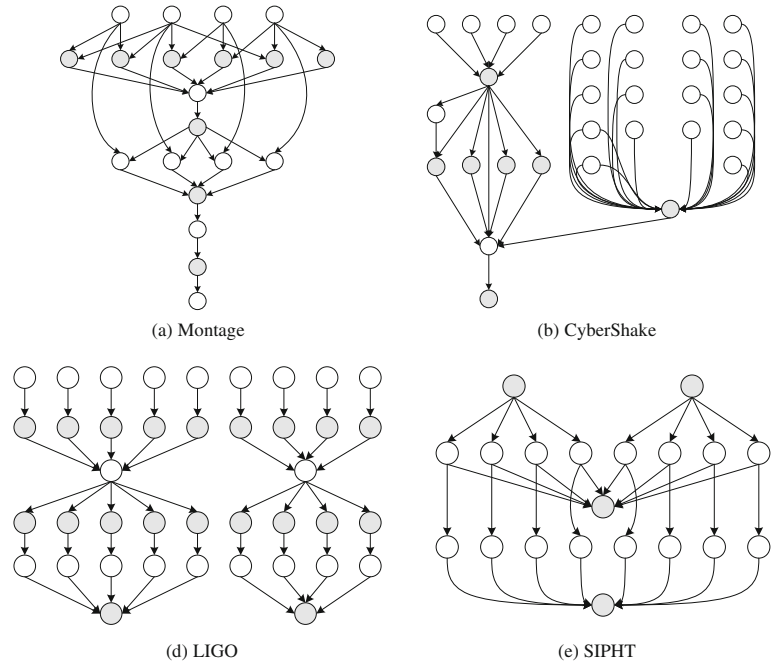

Fig. 4 Examples of workflow types

Table 7 Characteristics of workload A

Log name J_j	n	$\|V_j\|$	$\|E_j\|$	$\overline{p}_j$ (second)	p_j range (second)	$\overline{cpn}_j$	$\overline{size}_j$
Cybershake	20	200	392	226	[183–274]	4	99
Cybershake	20	100	192	222	[169–271]	4	49
Cybershake	20	50	92	224	[159–272]	4	24
Epigenomics	1	100	122	29878	29878	8	24
Epigenomics	1	46	54	7734	7734	9	10
Epigenomics	1	24	27	5586	5586	8	5
Gnome	20	200	244	20585	[9397–32434]	9	48
Gnome	20	100	122	16322	[7624–31212]	9	24
Gnome	20	50	54	20095	[9026–33091]	9	10
Inspiral	1	100	119	1336	1336	6	24
Inspiral	1	50	60	1415	1415	6	12
Inspiral	1	30	35	1337	1337	6	7
Ligo	20	200	241	1389	[1368–1408]	6	48
Ligo	20	100	120	1349	[1290–1411]	6	24
Ligo	20	50	60	1348	[1263–1396]	6	12
Montage	20	200	485	110	[101–123]	9	162
Montage	20	100	234	76	[74–79]	9	78
Montage	20	50	108	58	[56–61]	9	36
Sipht	20	200	224	4811	[4000–5467]	5	146
Sipht	20	100	112	4610	[4083–5374]	5	73
Sipht	20	50	58	4453	[3715–5368]	5	32

Table 8 Characteristics of workload B

Log name J_j	n	$\|V_j\|$	$\|E_j\|$	$\overline{p}_j$(second)	p_j range (second)	$\overline{cpn}_j$	$\overline{size}_j$
Cybershake	5	700	1388	252	[200,274]	4	348
Cybershake	5	600	1189	250	[215,275]	4	298
Cybershake	10	500	990	247	[179–276]	4	248
Cybershake	10	400	792	220	[160–266]	4	199
Cybershake	10	300	592	218	[188–270]	4	149
Cybershake	20	200	392	226	[183–274]	4	99
Cybershake	20	100	192	222	[169–271]	4	49
Cybershake	20	50	92	224	[159–272]	4	24
Epigenomics	1	100	122	29878	29878	8	24
Epigenomics	1	46	54	7734	7734	9	10
Epigenomics	1	24	27	5586	5586	8	5
Gnome	5	700	854	21276	[7728–30454]	9	168
Gnome	5	600	704	20751	[9515–30911]	9	135
Gnome	10	500	591	27527	[8526–35022]	9	114
Gnome	10	400	470	17505	[8858–26685]	9	90
Gnome	10	300	359	18423	[9214–34686]	9	70
Gnome	20	200	244	20585	[9397–32434]	9	48
Gnome	20	100	122	16322	[7624–31212]	9	24
Gnome	20	50	54	20095	[9026–33091]	9	10
Inspiral	1	100	119	1336	1336	6	24
Inspiral	1	50	60	1415	1415	6	12
Inspiral	1	30	35	1337	1337	6	7
Ligo	5	700	873	1407	[1394–1415]	6	174
Ligo	5	600	752	1407	[1400–1413]	6	149
Ligo	10	500	617	1404	[1370–1416]	6	123
Ligo	10	400	490	1404	[1390–1413]	6	97
Ligo	10	300	369	1394	[1379–1418]	6	73
Ligo	20	200	241	1389	[1368–1408]	6	48
Ligo	20	100	120	1349	[1290–1411]	6	24
Ligo	20	50	60	1348	[1263–1396]	6	12
Montage	5	700	1734	263	[236–298]	9	578
Montage	5	600	1485	238	[209–274]	9	495
Montage	10	500	1236	209	[186–237]	9	412
Montage	10	400	983	185	[158–196]	9	328
Montage	10	300	734	140	[129–161]	9	245
Montage	20	200	485	110	[101–123]	9	162
Montage	20	100	234	76	[74–79]	9	78
Montage	20	50	108	58	[56–61]	9	36
Sipht	5	700	792	5285	[5044–5473]	5	493
Sipht	5	600	676	5292	[5116–5515]	5	429
Sipht	10	500	564	5143	[4845–5367]	5	356
Sipht	10	400	452	5126	[4447–5602]	5	283
Sipht	10	300	340	4994	[4637–5553]	5	210
Sipht	20	200	224	4811	[4000–5467]	5	146
Sipht	20	100	112	4610	[4083–5374]	5	73
Sipht	1	60	68	4491	4491	5	42
Sipht	20	50	58	4453	[3715–5368]	5	32
Sipht	1	30	34	3399	3399	5	21

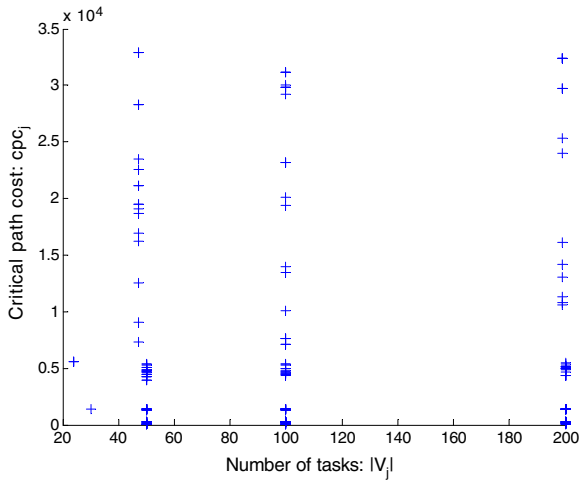

Fig. 5 Critical path cost versus number of tasks per workflow in workload *A*

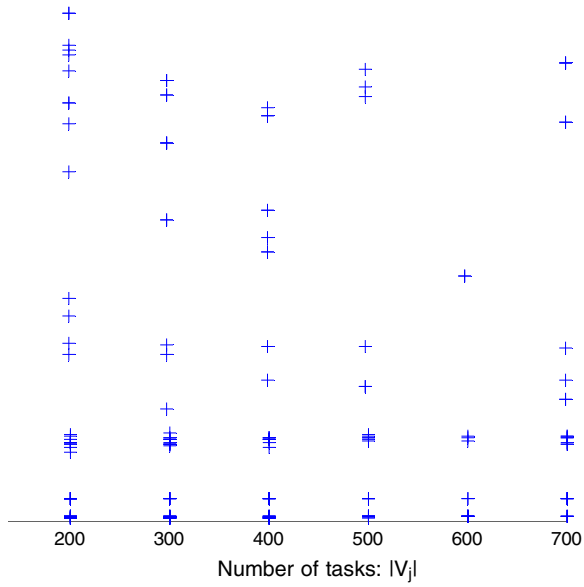

Fig. 7 Critical path cost and number of tasks per workflow in workload *B*

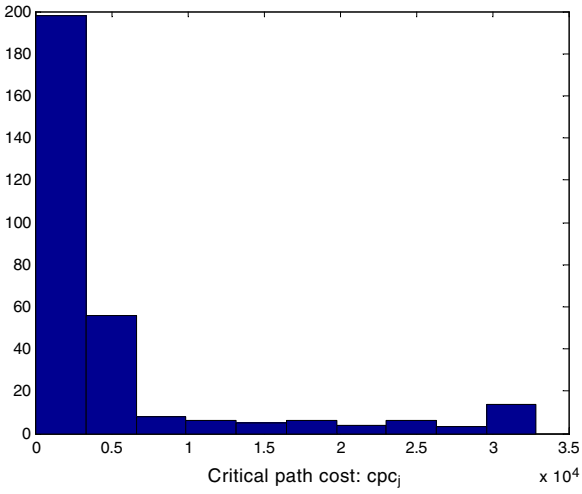

Fig. 6 Critical path cost histogram in workload *A*

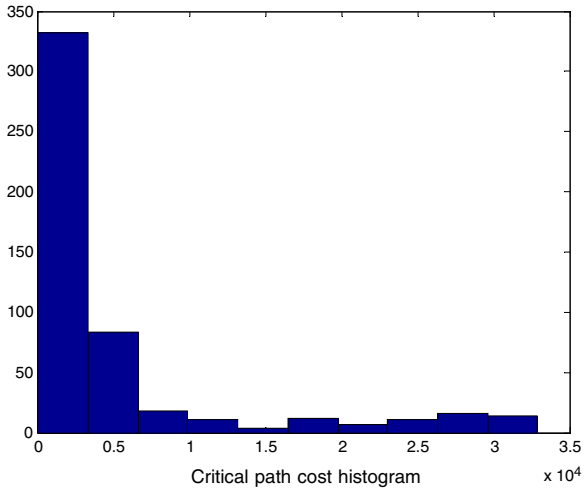

Fig. 8 Critical path cost histogram in workload *B*

A.2 Robustness

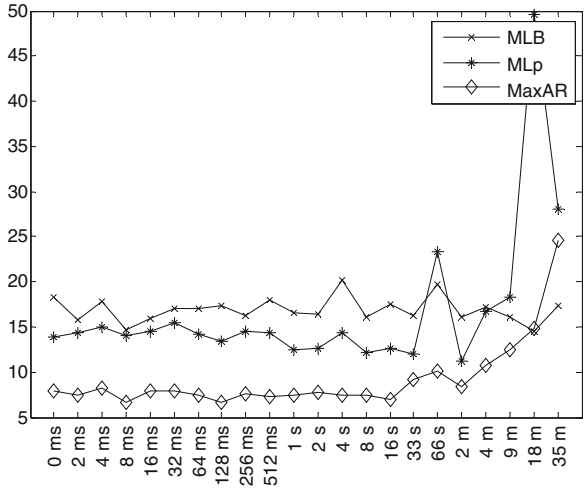

Fig. 9 Approximation factor performance degradation versus refresh interval

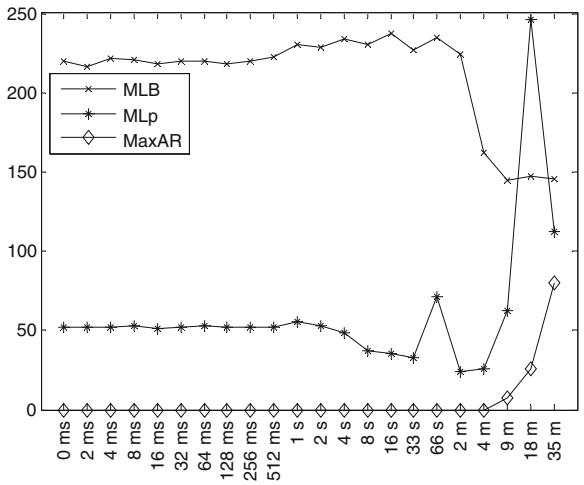

Fig. 11 Mean critical path waiting time performance degradation versus refresh interval

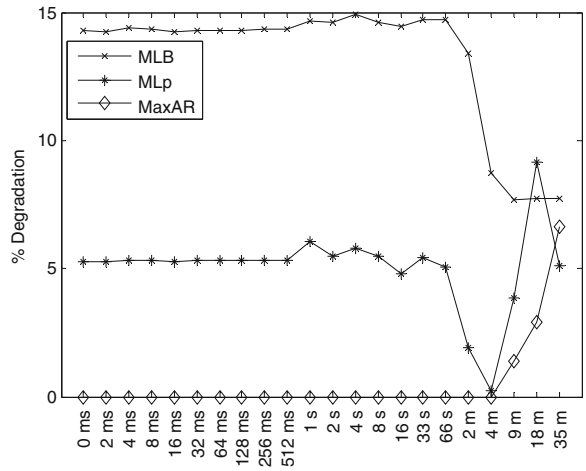

Fig. 10 Mean critical path slowdown performance degradation versus refresh interval

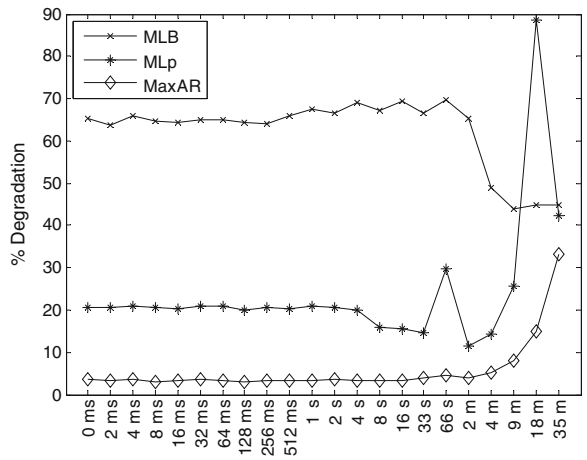

Fig. 12 Average performance degradation versus refresh interval

References

1. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer (2008)
2. Mccreary, C., Khan, A.A., Thompson, J.J., Mcardle, M.E.: A comparison of heuristics for scheduling dags on multiprocessors. In: International Parallel and Distributed Processing Symposium (IPPS94), pp. 446–451. Cancun, México (1994)
3. Kwong, K.Y., Ahmad, I.: Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Trans. Parallel Distrib. Syst. **7**, 506–521 (1996)
4. Kwok, Y.-K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4), 406–471 (1999)
5. Leung, J., Kelly, L., Anderson, J.H.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton (2004)
6. Rajakumar, S., Arunachalam, V.P., Selladurai, V.: Workflow balancing strategies in parallel machine scheduling. Int. J. Adv. Manuf. Technol. **23**, 366–374 (2004)
7. Wieczorek, M., Prodan, R., Fahringer, T.: Scheduling of scientific workflows in the askalon grid environment. SIGMOD Record **34**(3), 56–62 (2005)
8. Bittencourt, L.F., Madeira, E.R.M.: A dynamic approach for scheduling dependent tasks on the xavantes grid middleware. In: MCG'06: Proceedings of the 4th International Workshop on Middleware for Grid Computing. MCG'06, pp. 10–16. ACM, New York (2006)
9. Jia, Y., Rajkumar, B.: Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program. **14**(3), 217–230 (2006)
10. Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., Blackburn, K., Meyers, D., Samidi, M.: Scheduling data-intensive workflows onto storage-constrained distributed resources. In: CCGRID'07: Proceedings of the 7th IEEE Symposium on Cluster Computing and the Grid. CCGRID'07, pp. 14–17 (2007)
11. Szepieniec, T., Bubak, M.: Investigation of the dag eligible jobs maximization algorithm in a grid. In: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing, GRID'08, pp. 340–345. IEEE Computer Society, Washington (2008)
12. Singh, G., Su, M.-H., Vahi, K., Deelman, E., Berriman, B., Good, J., Katz, D.S., Mehta, G.: Workflow task clustering for best effort systems with Pegasus. In: MG'08: Proceedings of the 15th ACM Mardi Gras Conference, pp. 1–8. ACM, New York (2008)
13. Masko, L., Dutot, P.F., Mounie, G., Trystram, D., Tudruj, M.: Scheduling moldable tasks for dynamic SMP clusters in soc technology. In: Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol. 3911, pp. 879–887. Springer (2005)
14. Masko, L., Mounie, G., Trystram, D., Tudruj, M.: Program graph structuring for execution in dynamic SMP clusters using moldable tasks. In: International Symposium on Parallel Computing in Electrical Engineering, PAR ELEC 2006, pp. 95–100 (2006)
15. Singh, G., Kesselman, C., Deelman, E.: Optimizing grid-based workflow execution. J. Grid Computing **3**, 201–219 (2005)
16. Bittencourt, L.F., Madeira, E.R.M.: Towards the scheduling of multiple workflows on computational grids. J. Grid Computing **8**, 419–441 (2010)
17. Zhao, H., Sakellariou, R.: Scheduling multiple dags onto heterogeneous systems. In: Parallel and Distributed Processing Symposium, 20th International, IPDPS'06, p. 14. IEEE Computer Society, Washington (2006)
18. Topcuouglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
19. Sakellariou, R., Zhao, H.: A hybrid heuristic for dag scheduling on heterogeneous systems. In: 13th IEEE Heterogeneous Computing Workshop (HCW'04). IPDPS'04, pp. 111–123. IEEE Computer Society, Santa Fe (2004)
20. Zhu, L., Sun, Z., Guo, W., Jin, Y., Sun, W., Hu, W.: Dynamic multi dag scheduling algorithm for optical grid environment. Netw. Architect. Manag. Appl. V **6784**(1), 1122 (2007)
21. N'takpé, T., Suter, F.: Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. In: International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium, pp. 1–8 (2009)
22. Hsu, C.-C., Huang, K.-C., Wang, F.-J.: Online scheduling of workflow applications. In Grid environments. Future Gen. Comput. Syst. **27**, 860–870 (2011)
23. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**, 529–543 (2001)
24. Ramirez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., Gonzalez-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical Grids. J. Grid Computing **9**, 95–116 (2011)
25. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling parallel machines on-line. SIAM J. Comput. **24**, 1313–1331 (1995)
26. Condor high throughput computing. Available in: http://www.cs.wisc.edu/condor/. Cited August 2011
27. Openpbs. Available in: http://www.mcs.anl.gov/research/projects/openpbs/. Cited August 2011
28. Globus. Available in http://www.globus.org/. Cited August 2011
29. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: On-line hierarchical job scheduling on Grids with admissible allocation. J. Scheduling **13**, 545–552 (2010)
30. Workflow generator. Available in https://confluence.pegasus.isi.edu/display/pegasus/. Cited August 2010

31. Garey, M.R., Graham, R.L.: Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. **4**, 187–200 (1975)
32. Hirales-Carbajal, A., Tchernykh, A., Roblitz, T., Yahyapour, R.: A grid simulation framework to study advance scheduling strategies for complex workflow applications. In: IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–8 (2010)
33. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.-H., Vahi, K.: Characterization of scientific workflows. In: Third Workshop on Workflows in Support of Large-Scale Science, WORKS08, pp. 1–10 (2008)
34. Lee, C.B., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In: Job Scheduling Strategies for Parallel Processing, pp. 253–263 (2004)
35. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
36. Dolan, E.D., Moré, J.J., Munson, T.S.: Optimality measures for performance profiles. Siam. J. Optim. **16**, 891–909 (2006)
37. Dongarra, J.J., Jeannot, E., Saule, E., Shi, Z.: Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In: Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'07, pp. 280–288. ACM, New York (2007)
38. Saule, E., Trystram, D.: Analyzing scheduling with transient failures. Inform. Process. Lett. **109**(11), 539–542 (2009)
39. Canon, L.-C., Jeannot, E., Sakellariou, R., Zheng, W.: Comparative evaluation of the robustness of dag scheduling heuristics. In: Gorlatch, S., Fragopoulou, P., Priol, T. (eds.) Journal of Grid Computing, pp. 73–84. Springer, New York (2008)
40. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for scheduling parameter sweep applications in grid environments. In: Heterogeneous Computing Workshop, pp. 349–363 (2000)
41. 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**, 1175–1220 (2002)
42. Casanova, H.: SimGrid: a toolkit for the simulation of application scheduling. In: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 430–437 (2001)
43. Sulistio, A., Yeo, C.S., Buyya, R.A.: A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools. Software: Practice and Experience (SPE) **34**(7), 653–673 (2004). ISSN: 0038-0644