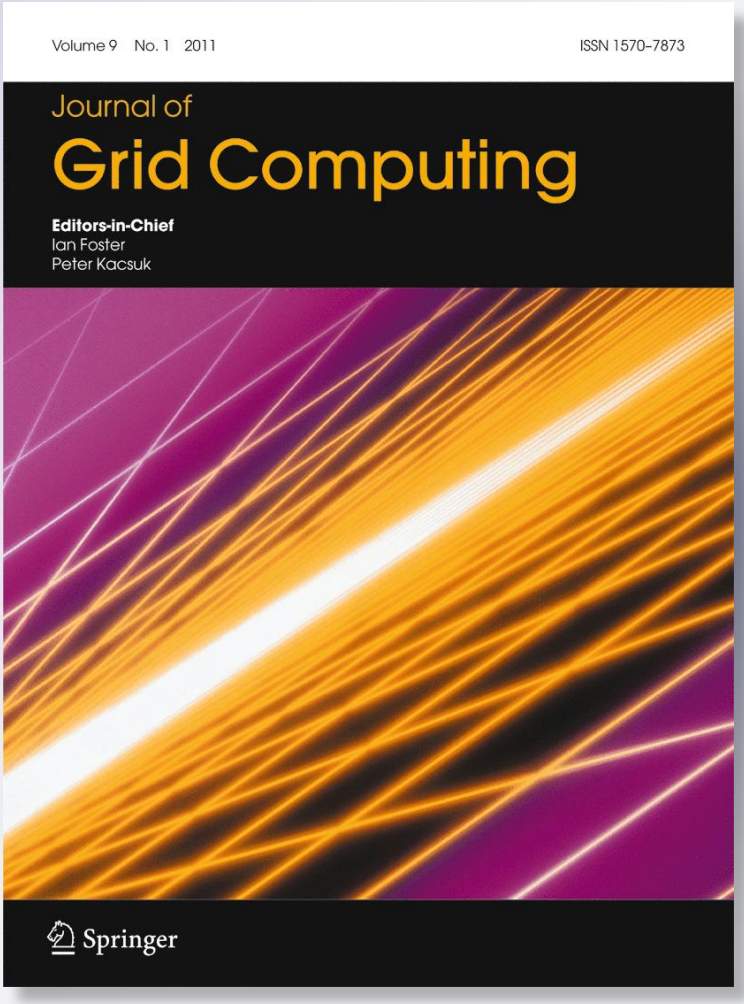
Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids

Journal of Grid Computing

ISSN 1570-7873
Volume 9
Number 1

J Grid Computing (2011)
9:95-116
DOI 10.1007/s10723-011-9179-y



Springer

J Grid Computing (2011) 9:95–116
DOI 10.1007/s10723-011-9179-y

Job Allocation Strategies with User Run Time Estimates for Online Scheduling in Hierarchical Grids

Juan Manuel Ramírez-Alcaraz · Andrei Tchernykh · Ramin Yahyapour · Uwe Schwiegelshohn · Ariel Quezada-Pina · José Luis González-García · Adán Hirales-Carbajal

Received: 28 September 2010 / Accepted: 24 January 2011 / Published online: 19 February 2011

Abstract We address non-preemptive non-clairvoyant online scheduling of parallel jobs on a Grid. We consider a Grid scheduling model with two stages. At the first stage, jobs are allocated to a suitable Grid site, while at the second stage, local scheduling is independently applied to each site. We analyze allocation strategies depending on the type and amount of information they require. We conduct a comprehensive performance evaluation study using simulation and demonstrate that our strategies perform well with respect to several metrics that reflect both user- and system-centric goals. Unfortunately, user run time estimates and information on local schedules does not help to significantly improve the outcome of the allocation strategies. When examining the overall Grid performance based on real data, we determined that an appropriate distribution of job processor requirements over the Grid has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules. In general, our experiments showed that rather simple schedulers with minimal information requirements can provide a good performance.

Keywords Grid computing · Online scheduling · Resource management · Job allocation

J. M. Ramírez-Alcaraz · A. Tchernykh (✉) · A. Quezada-Pina · J. L. González-García · A. Hirales-Carbajal
Computer Science Department,
CICESE Research Center, Ensenada, BC, México
e-mail: chernykh@cicese.mx

A. Quezada-Pina
e-mail: aquezada@cicese.mx

J. L. González-García
e-mail: jlgonzal@cicese.mx

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

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

R. Yahyapour
IT and Media Center,
Technische Universität Dortmund,
44221 Dortmund, Germany
e-mail: ramin.yahyapour@udo.edu

U. Schwiegelshohn
Robotics Research Institute,
Technische Universität Dortmund,
44221 Dortmund, Germany
e-mail: uwe.schwiegelshohn@udo.edu

A. Hirales-Carbajal
Science Faculty,
Autonomous University of Baja California,
Ensenada, 22860 BC, México

1 Introduction

Due to the size and dynamicity of Grids, there is a need for an automatic and efficient process to allocate computational jobs to available resources. Various scheduling systems have already been proposed and implemented in different types of Grids [1–8]. Most academic studies either propose a completely distributed resource management system, see, for instance, [9], or suggest a central scheduler, see [10, 11], while real installations favour a combination of decentralized and centralized structures, see [12]. Such combined systems have also been proposed using a hierarchical multilayer resource management, see [13–20]. The highest layer is often called a Grid-layer scheduler that typically has a general view of job requests while specific details on the state of the resources remain hidden from it. A local resource management system knows in detail the resource state and the jobs that are actually forwarded to it. In very large systems, additional layers may exist in between. Therefore, an efficient resource management system for Grids requires a suitable combination of scheduling algorithms that support such multilayer structures.

In this paper, we conduct a comprehensive performance evaluation study of a basic two layer online Grid scheduling model. The first layer allocates a job to a suitable parallel machine using a given selection criteria while the second layer applies potentially machine dependent scheduling algorithms to the allocated jobs. As Grid resources are typically connected by wide area networks and do not share the same management system, job migration between different resources requires a significant overhead and is technically challenging. Hence, we do not consider any multisite job execution nor any job migration after a job has been allocated to a machine; that is, an allocated job must be executed on the assigned machine. Taking into account the recent advance of parallel processing due to multicore architectures, we assume rigid parallel jobs; that is, the jobs have a given degree of parallelism and must be assigned exclusively to the specified number of processors or cores during their execution. While machines in real Grids often exhibit different forms of heterogeneity, like different hardware, operating system and software, we restrict ourselves to machines with a different number of the same processors or cores as architectures of individual cores and their clock frequency tend to be rather similar. Therefore, we believe that the focus in our model is reasonable and representative for real installations and applications.

After formally presenting our Grid scheduling model in Section 2, we introduce hierarchical job scheduling algorithms and classify them in Section 3. Next, we discuss related work in Section 4. Experimental setups are presented in Section 5, followed by experimental results in Section 6. Finally, we conclude with a summary and an outlook in Section 7.

2 Model

We address an online scheduling problem: n parallel jobs $J_1, J_2, ..., J_n$ must be scheduled on m parallel machines (sites) $N_1, N_2, ..., N_m$. Let m_i be the number of identical processors or cores of machine N_i. We denote the total number of processors belonging to machines from N_1 to N_m by $m_{1,m} = \sum_{i=1}^{m} m_i$. Assume without loss of generality that the machines are arranged in non-descending order of their numbers of processors, that is $m_1 \leq m_2 \leq ... \leq m_m$ holds.

Each job J_j is described by a tuple $(r_j, size_j, p_j, p'_j, p''_j)$: its release date $r_j \geq 0$, its size $1 \leq size_j \leq m_m$ also called its processor requirement or its degree of parallelism, its execution time p_j, its user run time estimate p'_j, and its system runtime prediction p''_j. The release date of a job is not available before the job is submitted, and its processing time is unknown until the job has completed its execution (non-clairvoyant case). System-generated prediction p''_j can be used to generate better schedules in some scheduling approaches like backfilling, see Tsafrir et al. [33], even if they are often very inaccurate.

Further, $w_j = p_j \cdot size_j$, $w'_j = p'_j \cdot size_j$, $w''_j = p''_j \cdot size_j$ are the work, estimated work, and predicted work of job J_j, respectively. At its release

date, a job must be immediately and irrevocably allocated to a single machine. However, we do not demand that specific processors are immediately assigned to a job at its release date as well; that is, the processor allocation of a job can be delayed until the required number of processors is actually available.

A machine must execute a job by exclusively allocating exactly $size_j$ processors for an uninterrupted period of time p_j to it. As we do not allow multisite execution and co-allocation of processors from different machines, a job J_j can only run on machine N_i if $size_j \leq m_j$ holds. We assume that the resources involved are stable and dedicated to Grid.

We use $g_j = i$ to denote that job J_j is allocated to machine N_i, while n_i is the number of jobs allocated to machine N_i. The completion time of job J_j of instance I in a schedule S is denoted by $c_j\ (S, I)$ while the makespan of a schedule S and instance I is $C_{\max}(S, I) = \max_{J_j}\{c_j(S, I)\}$. We use $C^*_{\max}(I)$ to specify the optimal makespan of instance I. Whenever it is possible without causing ambiguity, we will omit instance I and schedule S.

For our simulation experiments, we use twenty four metrics. However when applying a detailed analysis, we restricted ourselves to three well known performance metrics: Mean waiting time $t_w = \frac{1}{n}\sum_{j=1}^{n}(c_j - p_j - r_j)$; mean bounded slowdown $SD_b = \frac{1}{n}\sum_{j=1}^{n}\frac{c_j - r_j}{\max\{10, p_j\}}$; and sum of weighted completion times (total weighted completion time) with a specific weight selection $SWCT_w = \sum_{j=1}^{n} c_j \cdot w_j$. They are commonly used to express the objectives of different stakeholders of Grid scheduling (end-users, local resource providers, and Grid administrators).

To avoid the emphasis on very short jobs (e.g., with close to zero runtime) in the slowdown metric, a commonly used threshold of 10 seconds was applied.

Schwiegelshohn [21] suggested to use the well known sum of weighted completion times $SWCT_{weight}$ as a system-centric metric with the weight of job J_j being its resource consumption $w_j = size_j \cdot p_j$. He showed that this metric exhibits many properties that are similar to the properties of the makespan objective.

Furthermore, we use t_w over response (turnaround) time $TA = \frac{1}{n}\sum_{j=1}^{n}(c_j - r_j)$, and t_w over sum of job waiting times $SWT = \sum_{j=1}^{n}(c_j - p_j - r_j)$. Both metrics differ in constants regardless of the scheduler being used. In the first case, the constant is the average runtime of all jobs, and, in the second case, it is the factor $1/n$. Besides we use $SWCT_{weight}$ over the competitive factor, and resource-centric metrics like utilization $U = \sum_{j=1}^{n}\frac{p_j \cdot size_j}{C_{\max} \cdot m_{1,m}}$ and throughput $Th = \frac{n}{C_{\max}}$. There is a close relationship between these metrics due to $C^*_{\max}$, $\sum_{j=1}^{n}\frac{p_j \cdot size_j}{m_{1,m}}$, and n being constants for a given experiment; x percent reduction in $C_{\max}$ corresponds to $\frac{x}{100\% - x}$ percent increase in the utilization and throughput [21].

Note that in our evaluation of experiments, we use the lower bound of the optimal makespan $\breve{C}^*_{\max}$ instead of the optimal makespan with $C^*_{\max} \geq \breve{C}^*_{\max} = \max\left\{\max_j(r_j + p_j), \frac{\sum_{j=1}^{n} w_j}{m}\right\}$ as we are, in general, not able to determine the optimal makespan. We denote our Grid machine model by GP_m. Our scheduling problem is characterized as $GP_m|r_j, size_j|\{t_w, SD_b, SWCT_w\}$ using the three field notation $(\alpha|\beta|\gamma)$ introduced by Graham et al. [22]. This notation describes the fields machine environment (α), job characteristics (β), and objective function (γ). Finally, we use the notation *MPS* (Multiple Parallel Scheduling) to refer to our problem, while the notation *PS* (Parallel Scheduling) describes the parallel job scheduling on a single parallel machine.

3 Classification of Algorithms

We consider only algorithms that have no knowledge about jobs other than the number of unfinished jobs in the system, their processor requirements and user runtime estimates.

3.1 Allocation Strategies

As already discussed, our Grid scheduling algorithms can be split into a global allocation part and a local scheduling part. Hence, we regard

MPS as a two stage scheduling strategy: $MPS = MPS\_Alloc + PS$. At the first stage, we allocate a suitable machine for each job using a given selection criterion. At the second stage, the *PS* algorithm is applied to each machine independently for the jobs allocated during the previous stage. Note that our algorithms proceed on a job-by-job basis.

We distinguish allocation strategies depending on the type and amount of information they require. Note that the overall processor number and the processor numbers of the machines are

Table 1 Allocation strategies

Allocation strategy	Level	Description
Random	1	Randomly allocates jobs to the admissible site
MLp	1	Allocates job j to the site with least load per processor at time r_j : $\min_{i=1..m}\left(\frac{n_i}{m_i}\right)$ The motivation behind *MLp* is to balance the load between processors of the machines
MPL	1	Allocates job j to the site with least job processor requirements per processor at time r_j : $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k}{m_i}\right\}$ of each site. The intuition behind *MPL* is to keep all processors as busy as possible. One advantage of *MPL* is its simplicity. It does not take into account neither the job execution time nor execution time estimate
LBal_S	1	Allocates job j to the site with the least standard deviation of job processor requirements per processor (taking into account all sites) when job j is assigned to it. $\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(PL_i^q-\overline{PL}\right)^2}$, where $PL^q_{i=1..m}=\frac{1}{m_i}\sum_{g_k=i}\left(size_k+size_j^q\right)$ and $size_j^q$ is the size of job j added to site q (see [14])
LBal_T	2	Allocates job j to the site with the least standard deviation of job execution time requirements per processor (taking into account all sites) when job j is assigned to it $\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(T_i^q-\overline{T}\right)^2}$, where $T^q_{i=1..m}=\frac{1}{m_i}\sum_{g_k=i}\left(p_k+p_j^q\right)$ and p_j^q is the time of job j added to site q
LBal_W	2	Allocates job j to the site with the least standard deviation of job work requirements per processor (taking into account all sites) when job j is assigned to it. $\min_{q=1..m}\sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(W_i^q-\overline{W}\right)^2}$, where $W^q_{i=1..m}=\frac{1}{m_i}\sum_{g_k=i}\left(w_k+w_j^q\right)$ and w_j^q is the work of job j added to site q
MLB	2	Allocates job j to the site with least work per processor at time r_j : $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{size_k\cdot p'_k}{m_i}\right\}$ of each site
MCT	3	Allocates job j to the site with earliest Grid completion time $\min\left\{C^i_{\max}\right\}$, where $C^i_{\max}=\max_{g_k=i}(C^i_k)$, and C^i_k is the completion time of job J_k in the site i. This causes some tasks to be assigned to machines that do not have the minimum completion time for it. *MCT* attempts to minimize the total completion time
MWT	3	Allocates job j to the site with minimum average job waiting time $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{t^k_w}{n_i}\right\}$
MWWT_S	3	Allocates a job j to the site with minimum average job weighted waiting time $\min_{i=1..m}\left\{\sum_{g_k=i}\frac{t^k_w\cdot weight_k}{n_i}\right\}$, where $weight_k=\{size_k, p'_k, w_k\}$
MWWT_T		
MWWT_W		
MST	3	Allocates job j to the site with earliest start time for this job $\min_{i=1..m}\left\{s^i_j\right\}$. For homogeneous platforms *MST* assigns each task to the machine with the minimum expected completion time for that task (Earliest-Finish-Time)
MSWCT_W	3	Allocates a job j to the site with minimum sum of weighted (work) completion time $\min_{i=1..m}\left\{\sum_{g_k=i}c_k\cdot w_k\right\}$

known. Further, we distinguish three levels of additionally available information used for job allocation (see Table 1).

Level 1: Once a job has been submitted only its processor requirements become known. There is no information on the processing time of jobs. However, the algorithm may use information on the already allocated jobs to each machine.

Level 2: We have access to the information of Level 1 and know the job runtime estimate p'_j or system runtime prediction p''_j.

Level 3: We have access to the information of Level 2 and to all local schedules as well. The GRMS (Grid Resource Management and Brokering Service) has information on the local schedule of the machines, and may consider this when allocating a job to a machine.

We selected our heuristics to support job-by-job allocation and to cover a wide range of information available in the moment of the allocation.

3.2 Parallel Machine Scheduling

Once a job has been allocated to a parallel machine, the local resource management system (*LRMS*) of this machine will generate a schedule. Different scheduling algorithms may be used by the *LRMS*: Many real systems apply First-Come-First-Serve (*FCFS*) algorithm that schedules jobs in the order of their arrival times. Although this algorithm performs reasonably well in practice, it produces very bad results in the worst case. This is also true for various versions of backfilling that usually improve the performance of *FCFS* for real workloads. Alternatively, list scheduling guarantees a maximum deviation $2 - 1/m$ from the optimum for a parallel machine with m processors. No better result is possible for non-clairvoyant scheduling; see [23].

In the remaining parts of this paper, we assume that the *LRMS* uses the following online parallel scheduling algorithm: *BEFF*—First-Come First-Serve policy with EASY Backfilling. It is well known that EASY Backfilling outperforms the *FCFS* algorithm for real workloads. In order to apply EASY Backfilling, user estimated runtimes are used.

4 Related Work

4.1 Theoretical Results

Schwiegelshohn et al. [24] showed that there is no polynomial time algorithm that guarantees schedules with a competitive bound <2 for $GP_m|r_j, size_j|C_{\max}$ and all problem instances unless $P = NP$. Therefore, the multiprocessor list scheduling bound of $2 - 1/m$ for the offline case, see [25], as well as for online submission [23], does not apply to Grids. Even more, list scheduling cannot guarantee a constant competitive bound for all problem instances in the concurrent submission case [18].

Further, the performance of Garey and Graham's list scheduling algorithm is significantly worse in Grids than in multiprocessors, see [24]. Schwiegelshohn et al. [24] presented an online non-clairvoyant algorithm that guarantees to generate a schedule with completion time being within a constant ratio 5 of the optimal solution.

Tchernykh et al. [27] analyzed adaptive admissible job allocation strategies. The model covers the main properties of Grids, for instance, machines with different sizes, and non-clairvoyant parallel jobs. The competitive factor varies between 17 and infinity with change of the admissible factor. It was shown that the algorithm is beneficial under certain conditions and allows an efficient implementation in real systems. Furthermore, a dynamic and adaptive approach is presented which can cope with different workloads and Grid properties.

Tchernykh et al. [27] provided a competitive analysis for the heuristic that is based on the load balancing model of Bar-Noy, see [28]. They provided an algorithm for online clairvoyant makespan scheduling of parallel rigid jobs in a Grid with the competitive factor $2e + 1$.

More theoretical results are available for offline clairvoyant and non-clairvoyant problem. We just mention main results, without details because they are not addressed in our paper. In [29], a related clairvoyant problem was addressed. The authors model an offline system consisting of N clusters with exactly m identical processors each, and propose an algorithm with a guaranteed worst-case performance ratio on the global makespan equal to 4. This problem is a version of the *MSP* (Multiple Strip Packing) problem, where jobs are considered as rectangles and must be allocated without rotation on consecutive processors [30]. This problem is strongly NP-hard.

A more general case of the non-clairvoyant problem considering a collection of k parallel machines of different sizes and jobs that do not require more resources than available on the biggest machine ($size_j \leq m_m$) was considered in [18, 19]. The authors address the performance of various 2-stage algorithms with respect to the makespan objective. They present algorithms with a competitive factor of 10.

Schwiegelshohn et al. [24] presented an offline version of the online non-clairvoyant algorithm that guarantees a competitive factor of 3 for the Grid scenario, when migration between machines is allowed. Marin Bougeret et al. [31] consider the clairvoyant version of the same problem with jobs that do not require more resources than available on the smallest machine. The proposed scheduling algorithm achieves a 5/2 ratio.

4.2 Runtime Prediction Models

Users of large parallel computers are typically required to provide runtime estimates for submitted jobs. The reasons are twofold: jobs that violate their estimates are killed; and estimates are used in the local machines scheduling policies like backfilling and its variants. In the paper, we separate kill-time from the runtime prediction and system generation predictions. The issues of user runtime estimates have become the focus of intensive research. Tsafrir et al. [32] analyzed workload logs collected from parallel machines in production use, and showed that user runtime estimates are highly inaccurate even for a single machine. Estimates are inherently modal, because users tend to repeatedly use the same runtime limits (e.g. 10 min, 1 h, and so on). The situation becomes worse when several geographically distributed computers with unknown characteristics are used as the user may not have real information on application behavior on such systems.

The user run time estimate accuracy of job J_j is calculated as the ratio of the real runtime to the estimate p_j/p'_j. System-generated prediction accuracy is calculated as the ratio of the real runtime to the system-generated prediction p_j/p''_j. To avoid under prediction and over prediction compensation during averaging, we define according to Tsafrir et al. [33]: $accuracy = 1$, if $p'_j = p_j$; p_j/p'_j, if $p'_j > p_j$; p'_j/p_j, if $p'_j < p_j$.

The study of the impact of the user runtime estimates accuracy on the performance of the scheduling policies for parallel computers has been presented in several works (see [3, 32–42]). We mention here only few aspects of the problem like user runtime estimation models, the accuracy of the runtime estimates; the impact of the usage of prediction techniques in the backfilling policies; etc. User estimates may serve as kill times, while system predictions can be used for scheduling. Talby et al. [42] presented three prediction techniques. The first one is based on the entire user historical data of the workload, along with user runtime estimates. The second one uses historical data only, without user estimates. In the third one neither historical information nor estimates are available.

Historical data can improve estimates due to the fact that users tend to repeatedly execute the same programs. To increase accuracy of the prediction, the following actions can be taken: prediction correction, prediction fallback, and propagation (see [33]).

Tsafrir et al. [33] showed that there is no dominance of the specific history windows size. They selected the last two preceding jobs ($k = 2$) and showed that it results in significant improvement, both in the accuracy of the prediction itself and in the resulting performance of the backfilling. The results clearly show that for the backfilling policies recency is more important than similarity. It is

better to use the last job by the same user than to search for the most similar job.

Grid scheduling policies are also mainly based on the user runtime estimates. Moreover, current Grid resource management is based on the assumption that users provide accurate estimates of job runtimes. These estimates have a significant impact on particular Grid resources (for instance, advance reservation), and on overall Grid performance. However, due to the heterogeneity of computational resources in Grids, users do not have enough information to provide how long their jobs will run with adequate accuracy. The usage of job execution time prediction techniques in Grid scheduling policies instead of user estimates has a crucial relevance. However, predictions have not been incorporated into resource allocation strategies; that is, into identification of the set of resources that best match the job requirements.

In this paper, we use the history-based system-generated prediction models incorporated into the job allocation policies to study their impact on the overall two layer computational Grid performance.

5 Experimental Setup

Two fundamental issues have to be addressed for how to set up a simulation environment for performance evaluation. On the one hand, representative workload traces are needed to produce dependable results. On the other hand, a good testing environment should be set up to obtain reproducible and comparable results. In this section, we present most common Grid trace-based simulation setups with an emphasis on the mentioned two issues.

5.1 Grid Configuration

We consider two Grid scenarios for evaluation: Grid1 and Grid2. In Grid1, we considered the seven sites with total 4,442 processors as presented in the Table 2. In Grid2, we considered nine sites with a total of 2,194 processors (see Table 3). Respective logs are used to create the Grid log. Details of the log characteristics can again be found in the Parallel Workloads Archive(PWA) [43] and the Grid Workloads Archive(GWA) [44].

5.2 Workload

The accuracy of the evaluation highly relies upon workloads applied. For testing the job execution performance under a dedicated Grid environment, we use Grid workload based on real production traces. Carefully reconstructed traces from real supercomputers provide a very realistic job stream for simulation-based performance evaluation of Grid job scheduling algorithms. Background workload (locally generated jobs) that is an important issue in non-dedicated Grid environment is not addressed.

Logs from PWA and GWA are used. The premise of the integration of several logs of production machines into a Grid log is based on the following. Grid logs contain jobs submitted by users of different sites; Grid execution context could be composed by these sites. Unification of these sites into a Grid will trigger to merge users and their jobs. It should be mentioned that

Table 2 Grid1 characteristics

	Location	Procs	Log
1	KTH—Swedish Royal Institute of Technology	100	KTH-SP2–1996–2.swf, 28489 jobs, 204 users
2	SDSC-SP2—San Diego Supercenter SP2	128	SDSC-SP2–1998–3.1-cln.swf, 73496 jobs, 437 users
3	HPC2N—High Performance Computing Center North, Sweden	240	HPC2N-2002–1.1-cln.swf, 527371 jobs, 256 users
4	CTC—Cornell Theory Center	430	CTC-SP2–1996–2.1-cln.swf, 79302 jobs, 679 users
5	LANL—Los Alamos National Lab	1024	LANL-CM5–1994–3.1-cln.swf, 201387 jobs, 211 users
6	SDSC-BLUE—San Diego Supercenter Blue Gene	1152	SDSC-BLUE-2000–3.1-cln.swf, 250,440 jobs, 468 users
7	SDSC-DS—San Diego Supercenter Data Star	1368	SDSC-DS-2004–1-cln.swf, 96089 jobs, 460 users

Table 3 Grid2 characteristics

	Location	Procs	Log
1	DAS2—University of Amsterdam	64	Gwa-t-1-anon_jobs-reduced.swf, 1124772 jobs, 333 users
2	DAS2—Delft University of Technology	64	
3	DAS2—Utrecht University	64	
4	DAS2—Leiden University	64	
5	KTH—Swedish Royal Institute of Technology	100	KTH-SP2–1996–2.swf, 28489 jobs, 204 users
6	DAS2—Vrije University Amsterdam	144	Gwa-t-1-anon_jobs-reduced.swf (cont.)
7	HPC2N—High Performance Computing Center North, Sweden	240	HPC2N-2002–1.1-cln.swf, 527371 jobs, 256 users
8	CTC—Cornell Theory Center	430	CTC-SP2–1996–2.1-cln.swf, 79302 jobs, 679 users
9	LANL—Los Alamos National Lab	1024	LANL-CM5–1994–3.1-cln.swf, 201387 jobs, 211 users

merging several independent logs to simulate a computational Grid workload does not guarantee representation of the real Grid with the same machines and users. For instance, if the site becomes a part of the computational Grid where bigger machines are available, users might submit bigger jobs not represented in the original log. Nevertheless, it is a good starting point to evaluate Grid scheduling strategies based on real logs in the case of the lack of publicly available Grid workloads. Time-zone normalization, profiled time intervals normalization, and invalid jobs filtering are considered. Several filters are applied to remove certain jobs: job number ≤ 0; submit time < 0; runtime ≤ 0; number of allocated processors ≤ 0; requested time ≤ 0; user ID ≤ 0; status $= 0, 4, 5$ ($0 =$ job failed; $4 =$ partial execution, job failed; $5 =$ job was cancelled, either before starting or during run). Detailed characteristics of individual logs can be found in the Parallel Workloads Archive and the Grid Workloads Archive.

It is well known that the demand of jobs is not equally distributed over the time and varies with the time of the day and the day of the week. Moreover, each individual log shows a different distribution. In addition, machines of the Grids are located in different time zones. This is a reason of the mentioned normalization of the used workloads as shifting the workloads by a certain time interval to represent more realistic setup. The jobs' demand in the resulting log is now aligned over the time intervals. We transformed the workloads so that all traces begin at the same weekday and at the same time of day. To this end, we removed all jobs until the first Monday at midnight. Note that the alignment is related to the local time, hence the time differences corresponding to the original time zones are maintained. Note also that this modification results in a loss of jobs within each workload.

Figures 9, 10, 11, 12, and 13 in the Appendix show details of the traces of the Grid1 and Grid2 used in our study. Figure 9 in the Appendix shows the number of jobs per week. Figure 10 in the Appendix shows the mean resource consumption per month. We can see that even the resource consumption demand is not equally distributed over the time in the original traces; it is more balanced in the Grid logs. Note that the number of jobs in Grid1 is just half of the number of jobs in Grid2. However, resource consumption is twice as much (Fig. 10 in the Appendix). It gives us two different scenarios for simulation. We have observed predominance of low parallel jobs in both logs (Figs. 11 and 12 in the Appendix). Maximal job sizes are 1368 and 1024 in Grid1 and Grid2, respectively. In Grid1, most of the jobs require at most 128 processors (95%). 1 processor is required by 18% of jobs; 8, 16, 32, 64 processors are required by 20%, 9%, 14%, and 8% of jobs, respectively. The workload in Grid2 is less parallel. Most of the jobs require at most 32 processors (97%), 1 processor is required by 36% of jobs; 2, and 4 processors are required by 26% and 9% of jobs, respectively. Figure 13 shows that users from *HPC2N* (IDs 674–930) and from *KTH-SP2* (IDs 931–1130) that participate in both Grids have worst accuracy of run time estimates.

6 Simulation Results

In this section, we analyze 14 allocation strategies together with EASY backfilling local scheduling algorithm.

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, often conflicting, performance goals: from minimizing response time to optimizing the resources utilization. Grid resource management involves multiple objectives and may use multi-criteria decision support. General multi-criteria decision methodology based on the Pareto optimality can be applied. 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 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. A 10% deviation for sum of completion time is very different to 10% in the competitive factor etc. This can be done by a definition of criteria weights or criteria ranking by their importance.

In order to provide effective guidance in choosing the best strategy, we performed a joint analysis of several metrics according to methodology proposed in Tsafrir et al. [33]. They introduce 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 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 three metrics. This is done relative to the best performing strategy for the metric, as follows: 100* strategy_metric/best_metric-100. Thus, each strategy is now characterized by three (six for two Grids) numbers, reflecting its relative performance degradation under the test cases. In the second step, we average these three values (assuming equal importance of each metric), and rank the strategies for each Grid. The best strategy, with the lowest average performance degradation, has rank 1; the worst strategy has rank 14.

Then we calculate the average performance degradation and ranking for Grid1 and Grid2. We distinguish overall performance degradation from individual performance in the two different Grid setups. Note that we try to identify strategies which perform reliable 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.

6.1 Allocation Strategies

Table 6 in Appendix shows results of the comprehensive and extensive simulation of 14 allocation strategies with *BEFF* local scheduling algorithm over 24 metrics, all test cases, Grid1 and Grid2 scenarios average. Rounded percentages of the performance degradations are presented for each strategy and metric. The value zero denotes the best strategy under the metric. Results are obtained under different conditions and test cases. Conditions in Grid1 are characterized by seven large sites with 634 processors per site in average, and 22,770 jobs per site in average. Conditions in Grid2 are characterized by nine low parallel sites with 243 processors per site in average, and 47,770 jobs per site in average.

For a more detailed analysis, Table 4 shows the performance degradation of 14 strategies for three considered metrics in Grid1 and Grid2 scenarios. For the strategies that were compared, the waiting times, slowdown, and total weighted completion times vary greatly. The difference between strategies is more than 136,000% for t_w, 75,000% for SD_b, and 28% for $SWCT_w$. Such big deviations show that these metrics are very sensitive to the allocation strategies. Their careful selection is needed in real systems.

Table 4 Percentages of the performance degradations of the strategies for Grid1 and Grid2

Strategy	Metric				
	Grid	t_w	SD_b	$SWCT_w$	Mean
MCT	1	1256	1284	0	847
	2	4833	4001	0	2945
MLB	1	463	408	0	290
	2	1799	754	0	851
LBal_S	1	11	18	0	10
	2	70	17	0	29
LBal_T	1	136322	75442	28	70597
	2	4205	2569	0	2258
LBal_W	1	1006	899	0	635
	2	5743	3346	0	3030
MLp	1	119	71	0	63
	2	153	53	0	69
MPL	1	0	0	0	0
	2	0	0	0	0
Random	1	51929	41924	3	31285
	2	18027	12196	0	10074
MST	1	80	91	0	57
	2	252	111	0	121
MSWCT_W	1	70233	47821	2	39352
	2	6515	3858	0	3458
MWWT_S	1	17303	12576	1	9960
	2	9132	7550	0	5561
MWT	1	252	204	0	152
	2	933	628	0	520
MWWT_T	1	8073	7382	1	5152
	2	30822	27638	0	19487
MWWT_W	1	15506	12889	1	9465
	2	51768	68410	0	40059

Figures 1, 2, and 3 show the ranking of the best six strategies according to the performance degradation for t_w, SD_b, and $SWCT_w$. Figure 4 shows the mean degradations of the strategies in Grid1 and Grid2 when considering all metrics average. Figure 5 shows the mean performance degradation ranking for all test cases.

As expected, *MCT* and *MST* are more efficient than other strategies to minimize the $C_{\max}$ criterion for both Grids. Moreover, for these test cases, the differences between *MLB, LBal_S, MLp, MPL*, and *MWT* are negligible. They are performing as good as *MCT* and *MST*.

The difference between strategies under the sum of weighted completion times metric is less than 0.04%. This is probably the case because in the on-line scenario the load can be light in the end of simulation period (after release of the last job); hence the completion times of the last jobs determine the makespan.

From observing Figs. 1, 2, and 3, we find that the *MPL* and *LBal_S* allocation strategies taking only job sizes into account perform better than the other algorithms for waiting time, bounded slowdown, and total weighted completion time metrics. The difference in the relative performance is 19% when considering all metrics average.

The same result is observed over a wide range of simulation parameters. More information available in the moment of allocation such as job runtime estimates, and local schedules did not help to construct better Grid schedules. One of the reasons is that the user time estimates are known to be inaccurate. The *MLp* and *MST* heuristics also performed well for considered cases, giving the second best results. They are 69.67% and 90.43% worse than *MPL* when considering all metrics average.

Algorithms *MWT* and *MLB* are more than 300% worse than *MPL* when considering all metrics average.

The use of the weighted function in allocation strategies (*LBal_T, LBal_W, MWWT_T, MWWT_W, MWWT_S, MSWCT_W*) with execution time p'_j and resource consumption w_j as a weight shows a decrease in performance as to be expected for non weighted metrics. In contrast to *MPL* and *LBal_S*, they always performed poorly, ranking only on 9th–14th position.

Results indicate that the average relative performance of the *MPL* and *LBal_S* allocation strategies does not depend significantly on the workload used, Grid configuration, and on the performance metric. It turns out that *MPL* and *LBal_S* are the best performing algorithms. Besides the performance aspect, the use of *MPL* and *LBal_S* does not requires additional management overhead such as requesting info about local schedules or constructing preliminary schedules by the broker.

From Table 4 and Table 6 in the Appendix, we can see that the results vary greatly. However, they clearly indicate that *MPL* and *LBal_S* are able to achieve better performance than other algorithms in almost all test cases. Note that the advantage of these algorithms is more

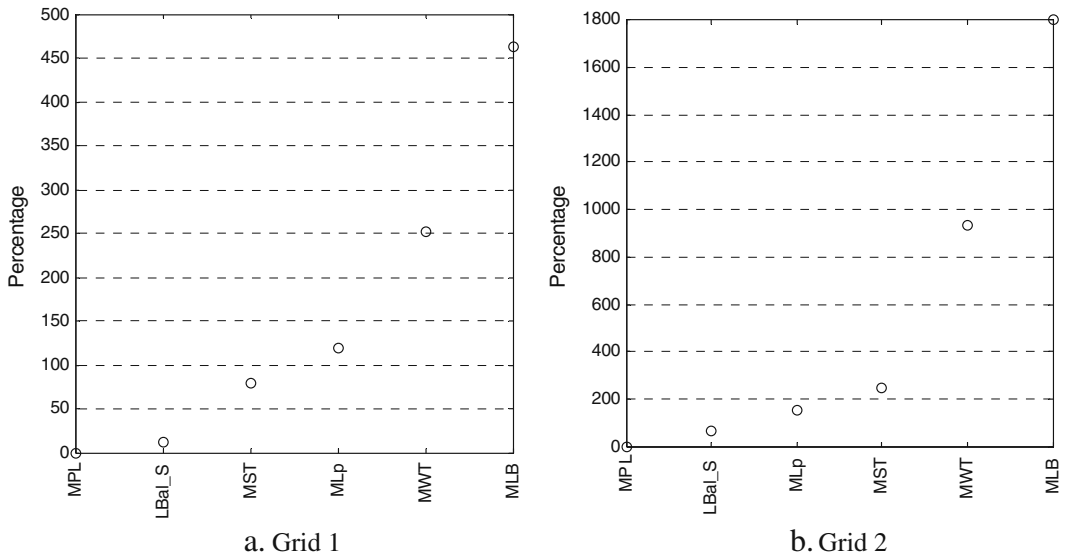


Fig. 1 t_w degradation ranking of best six strategies

significant when the traditional metrics are considered. When the deviation of the sizes, time and work criteria are considered ($LBal_s$, $LBal_t$, $LBal_w$), there exists a small number of others efficient allocations. We conclude that the *MPL* and *LBal_S* strategies are robust and stable even in significantly different conditions. *MLp* and *MST* also provide minor performance degradation and are able to cope with different demands.

Obtained results help us to select the appropriate strategy for real life Grids that depends on preferences of different decision makers of Grids (end-users, local resource providers, and Grid administrators), and an importance of every criterion or a relative importance between criteria.

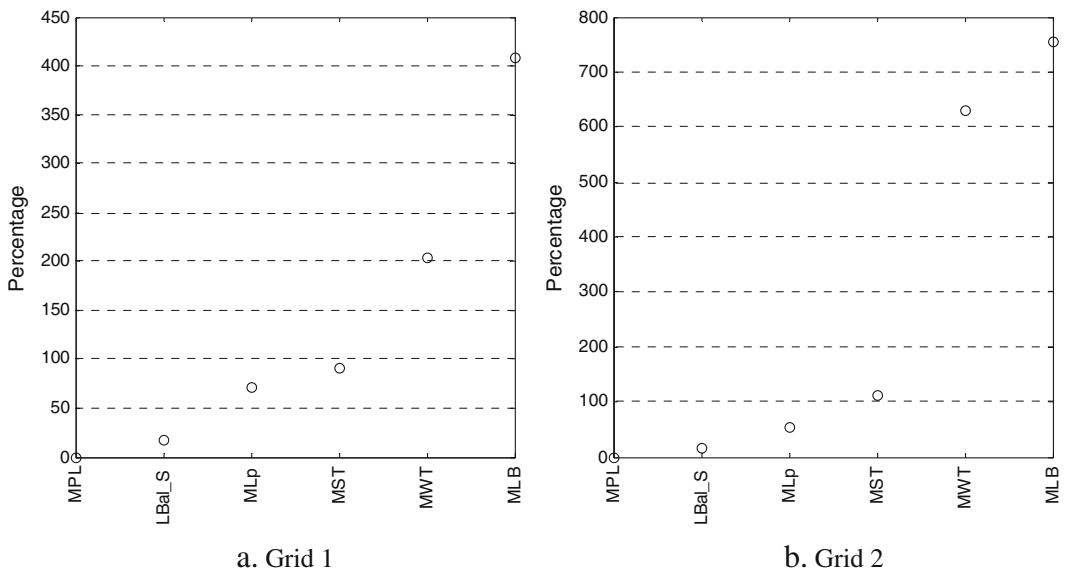
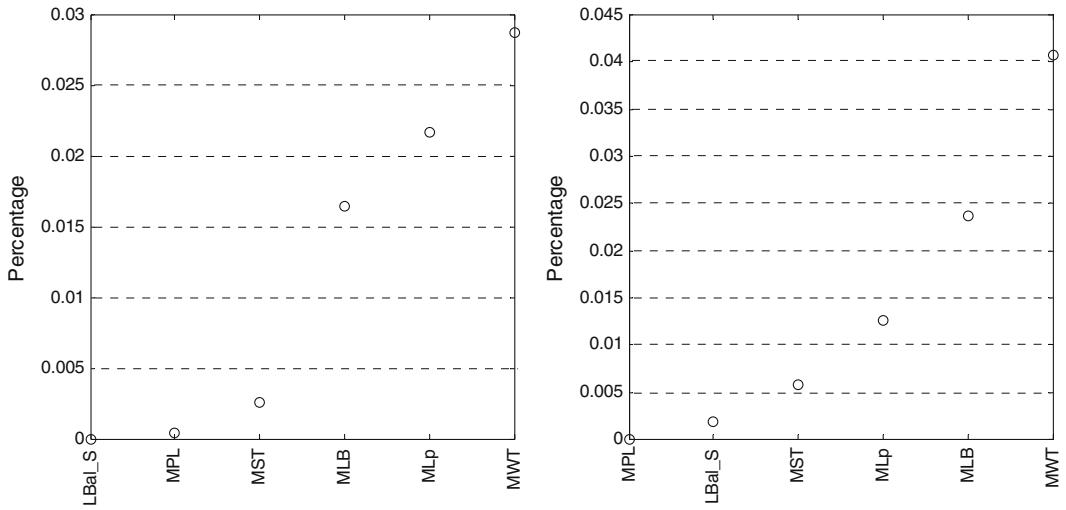


Fig. 2 SD_b degradation ranking of best six strategies



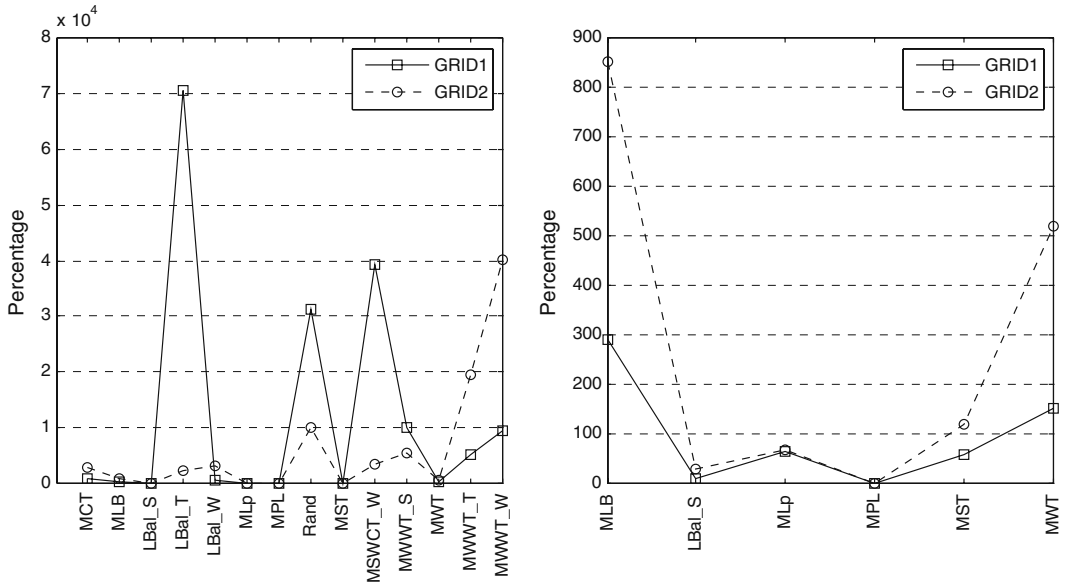
a. Grid 1 a. Grid 2

Fig. 3 $SWCT_w$ degradation ranking of best six strategies

6.2 Allocation Strategies with Historical System-Generated Run Time Prediction

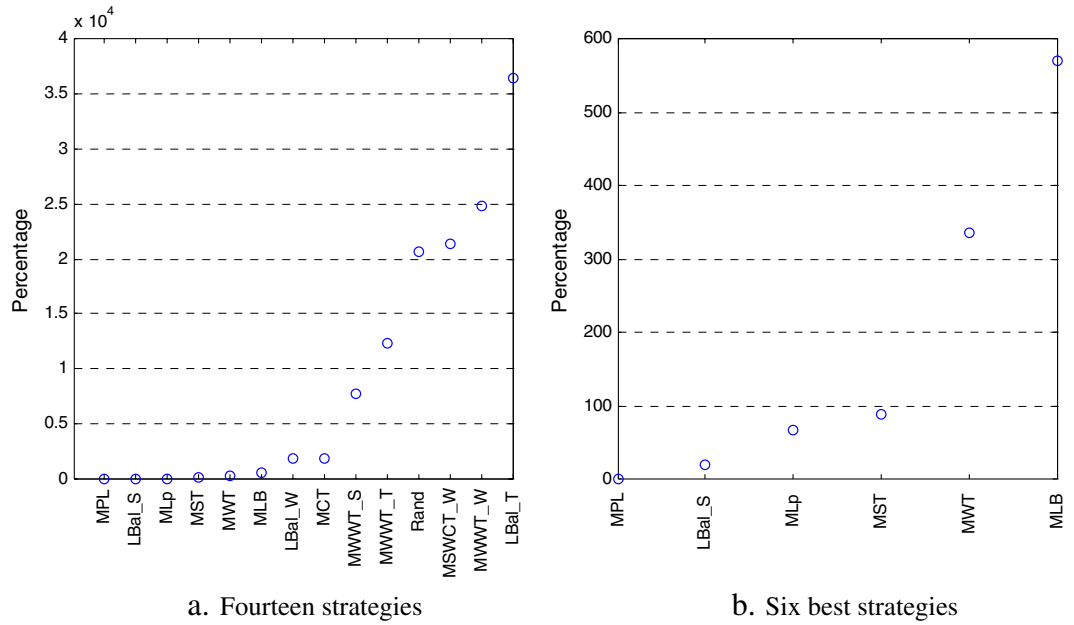
In this section, we present results of experimental analysis of the impact of system-generated predictions incorporated into *MST* and *MWT* allocation strategies on the overall Grid performance.

First, we considered the history-based system-generated prediction models. For each allocation strategy, we calculate three metrics with varying history window size from 1 to 10 for each of eight prediction models (Table 5). Then, we evaluate the degradation in performance of each prediction model and each window size under each metric.



a. Fourteen strategies b. Six best strategies

Fig. 4 Mean degradation of allocation strategies in Grid1 and Grid2



a. Fourteen strategies b. Six best strategies

Fig. 5 Mean degradation ranking of all test cases

This is done relative to the ideal case with system-generated prediction accuracy being equal to 1 ($p''_j = p_j$). Thus, each prediction model is now characterized by 30 parameters, reflecting its relative performance degradation under the test cases.

We use a well-known approach where prediction is calculated as the average runtime of selected preceding jobs. For a newly submitted job j, the average runtime of the preceding and already completed jobs i of the same user is calculated as

Table 5 System-generated prediction models

Prediction model	Name	Description
Hrecent_k	M1	k recent jobs
Hrecent_k_p′	M2	k recent jobs with the same user runtime estimate $(p'_i = p'_j)$
Hrecent_k_size	M3	k recent jobs with the same size ($size_i = size_j$)
Hrecent_k_size& p′	M4	k recent jobs with the same $p'_i = p'_j$ and $size_i = size_j$
Hk	M5	at most k jobs
Hk_p′	M6	at most k jobs with the same user runtime estimate $(p'_i = p'_j)$
Hk_size	M7	at most k jobs with the same size ($size_i = size_j$)
Hk_size& p′	M8	at most k jobs with the same $p'_i = p'_j$ and $size_i = size_j$
C1-model	M9	(Constant) The system prediction (p''_j) is calculated by multiplying the user runtime estimate by a constant. $p''_j = c_1 \cdot p'_j$ where c_1 is the mean accuracy in the available log
C2-model	M10	(Constant) The system prediction (p''_j) is calculated by multiplying the user runtime estimate by a constant. $p''_j = c_2 \cdot p'_j$ where c_2 is a predefined constant from 0.01, 0.02,...0.99, 1 (from 1% to 100% of the user runtime estimate)
U1-model	M11	(Uniform distribution) The system prediction is uniformly distributed in $p''_j = U[L_1, R_1] \cdot p'_j$, where $L_1 = \min_j \{p_j/p'_j\}$, and $R_1 = \max_j \{p_j/p'_j\}$ in the available log. Accurate user run time estimate corresponds to $L_1 = 1$ and $R_1 = 1$
U2-model	M12	(Uniform distribution) The system prediction is uniformly distributed in $p''_j = U[L_2, R_2] \cdot p'_j$, where L_2 is the average of the half smallest values $\{p_j/p'_j\}$ in the available log and R_2 is the average of the half biggest values $\{p_j/p'_j\}$ in the log. Accurate user run time estimate corresponds to $L_2 = 1$ and $R_2 = 1$

$p''_j = 1/k \sum_i^k p_i$, where k is the size of the history window which determines the set of previous jobs used for prediction.

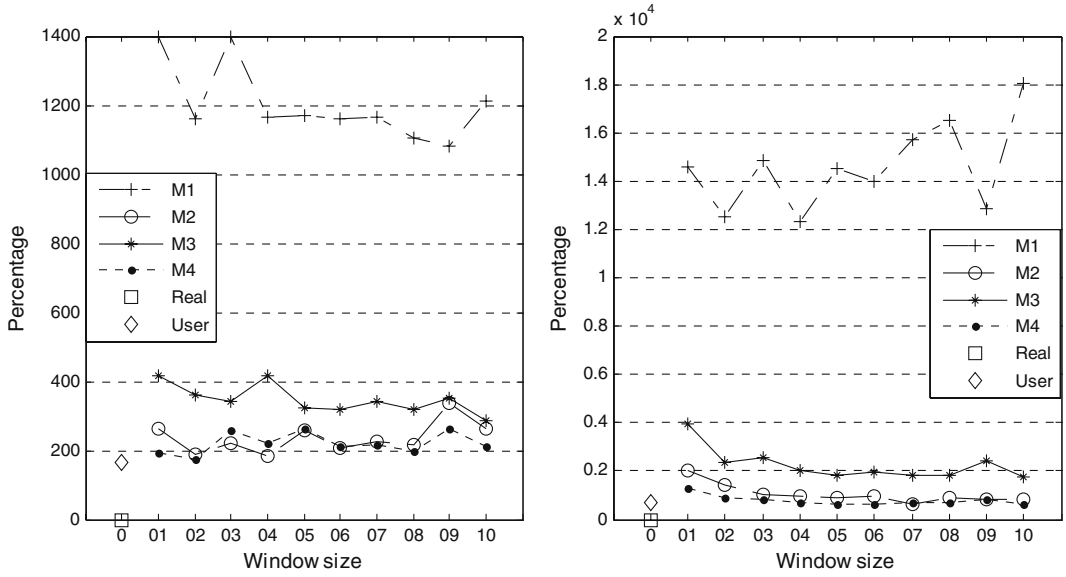
In the *Hrecent* models if k jobs that satisfy parameters of the model are not available, the initial user runtime estimate is used $p''_j = p'_j$ (see [33]). The algorithm only uses jobs that match the given criterion to generate a prediction (Table 5).

In the *Hk* group of models, at most k preceding jobs are selected for prediction. If k jobs that satisfy parameters of the model are not available, the average runtime of the eligible preceding jobs is calculated instead. For instance, for $k = 3$, if only two jobs are available in the entire user history, these two jobs are used to calculate the average, if only one job i is available, $p''_j = p_i$.

Figures 6 and 7 show mean degradation of *MST* and *MWT* over three metrics average varying history windows size for Grid1 and Grid2 scenarios. Figure 8 presents mean degradation ranking of all test cases. Results of the use of four mentioned system-generated run-time prediction models, of user runtime estimate, and of job runtime (as a reference point) are presented.

For *MST* (Fig. 6), we observe that *Hrecent_k_p′*, *Hrecent_k_size& p′* show best results in both Grids with small deviations of the degradation performance. For *MWT* (Fig. 7), there is no dominance of the specific model. However, the large history window size (8–10) improves results of *Hrecent_k_size& p′* model that consider the history of most similar jobs. Results of *MWT* also show that there is no improvement in the resulting performance of the *MWT*.

The results of the comprehensive and extensive simulation clearly indicate that it is better to use the most similar jobs characterized by the same user estimate and size (*size& p′*) as these models dominate in all test cases. Increasing the history window size slightly improve results. This is probably so because using the larger window gives more chances to find similar jobs. However, there is no dominance of the specific windows size. Using an average over the last one to ten preceding jobs show worst results in all test cases. Figure 8 indicates that there is no improvement in the performance of the *MST* and *MWT* allocation strategies with history-based system-generated predictions over the strategy that uses user run time estimates. This qualitative conclusion does not depend significantly on the



a. Grid 1. b. Grid 2.

Fig. 6 Mean degradation versus the history window size. *MST* allocation strategy

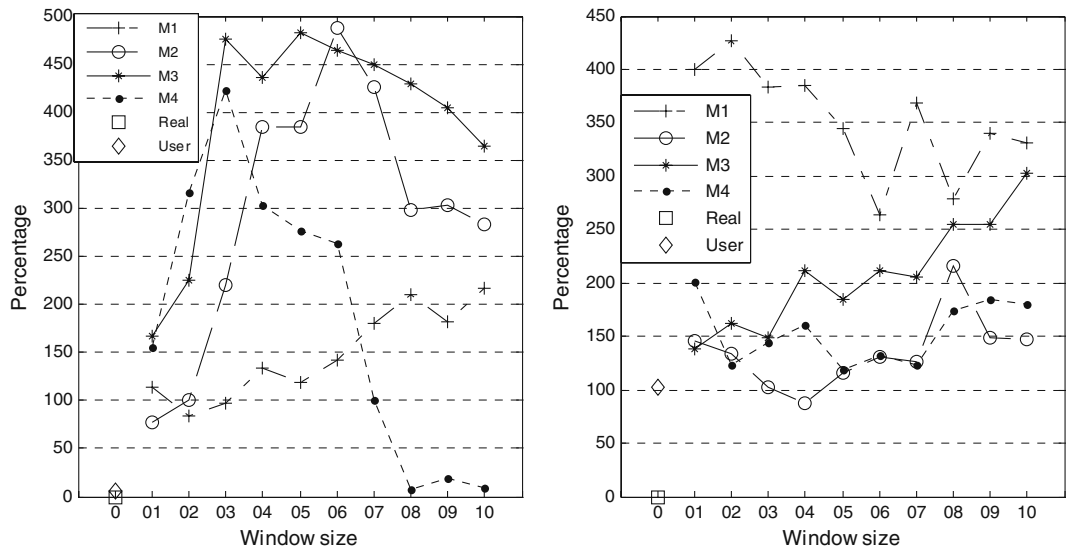


Fig. 7 Mean degradation versus the history window size. *MWT* allocation strategy

workload used, and on the performance metric. It turns out that the history-based system-generated prediction model incorporated into the job allocation policies and local scheduling algorithms does not provide as good results for multisite execution as for backfilling scheduling policies in the single site. We consider system-generated predictions that are calculated once in the release time with limited history information available. Another observation is that jobs allocated to

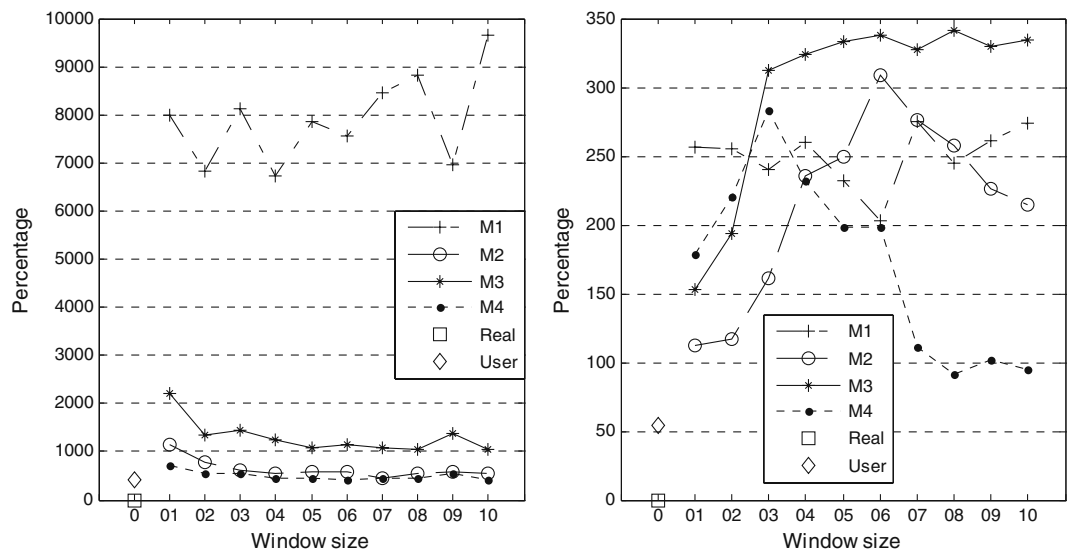


Fig. 8 Mean degradation of all test cases

the specific site could reduce an improvement of the system-generated predictions for backfilling. For instance, in the scenario with Grid sites of different sizes, the good job allocation does not create situations in which large machines are occupied by jobs with small processor requirements causing highly parallel jobs to wait for their execution (see [27]). However, jobs with small processor requirements and execution times (user estimates) are necessary for effective backfilling. The distribution of jobs in a two layer hierarchical online Grid scheduling model could also limit backfilling in the local resources. Each local site has a reduced number of jobs comparing to a one site Grid (one machine backfilling considered in [26]). This is limited to our Grid scenarios: if considering a smaller number of sites, system-generated prediction yields better performance results. Using a shortest job backfilled first (SJBF) backfilling order by the local schedulers could leads to a performance improvement [26].

We also analysed models that are based on the premise that the previous logs are available and can be analyzed for learning and improving the prediction parameters (*C1-model, C2-model, U1-model, and U2-model*) (Table 5). Results of *Hk*, *C1*, *C2*, *U1*, and *U*2 models sometimes show comparable or a bit better performance, but cannot be used to consistently obtain superior performance than *Hrecent* models. Obtained results for these models are not presented here.

7 Conclusion and Future Work

As Grids become more prevalent, techniques for efficiently utilizing their resources become increasingly significant. The problem of resource allocation and scheduling is crucial not only to achieve high Grid performance, but also to satisfy various user demands in an equitable fashion. While theoretical worst case Grid scheduling models are beginning to emerge, fast statistical techniques applied to real data have empirically been shown to be effective.

In this paper, we address non-preemptive non-clairvoyant online scheduling of parallel jobs on a Grid. We present a detailed performance evaluation of job allocations algorithms, covering aspects like performance and robustness. Our extensive study results in several contributions. Firstly, we identify several levels of information available to make scheduling decisions with respect to job allocations. We discussed and analyzed fourteen allocation strategies depending on the type and amount of information they require together with EASY backfilling local scheduling algorithm.

The selection of the appropriate strategy for Grids depends on preferences of different decision makers of Grids (end-users, local resource providers, and Grid administrators), and an importance of every criterion or a relative importance between criteria. To provide effective guidance in choosing a good strategy we performed a joint analysis of three metrics based on the degradation in performance of each strategy under each metric.

Simulation results presented in the paper reveal that in terms of minimizing waiting time, slowdown, total weighted completion time, and their degradation, average *MPL* and *LBal_S* allocation strategies outperform the other algorithms. They dominate in almost all test cases. We conclude that the strategies are robust and stable even in significantly different conditions. *MLp* and *MST* also provide minor performance degradation and able to cope with different demands.

We find that the information about user run time estimate and local schedules does not help to improve significantly allocation strategies. When examining the overall Grid performance on the real data, we determined that appropriate distribution of job processor requirements over the Grid has a higher performance than an allocation of jobs based on user run time estimates and information on local schedules.

The end result suggests simple schedulers, which require minimal information and little computational complexity; nevertheless, they achieve significant improvements in performance.

Appendix

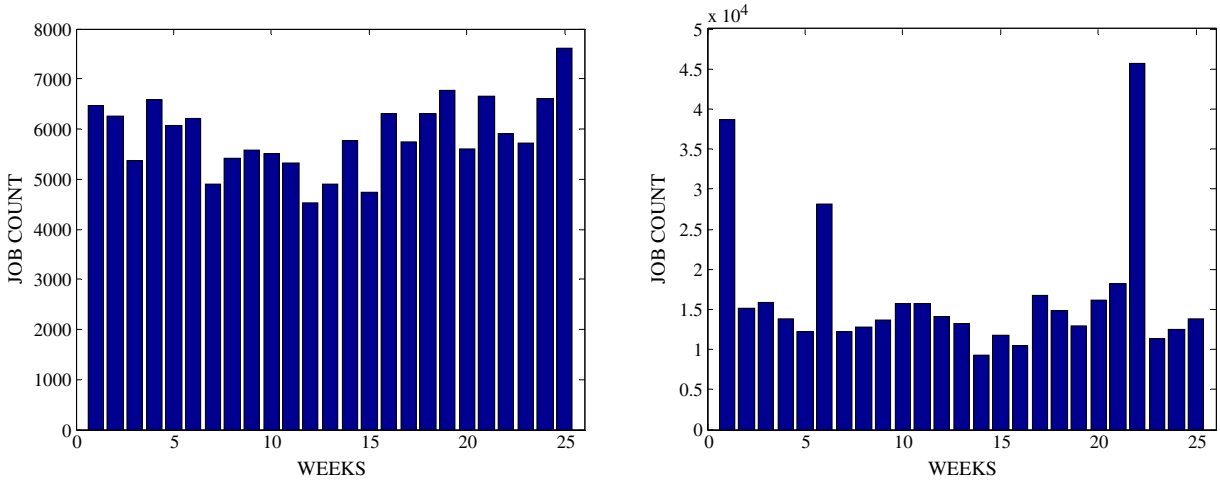


Fig. 9 Number of jobs in simulated periods

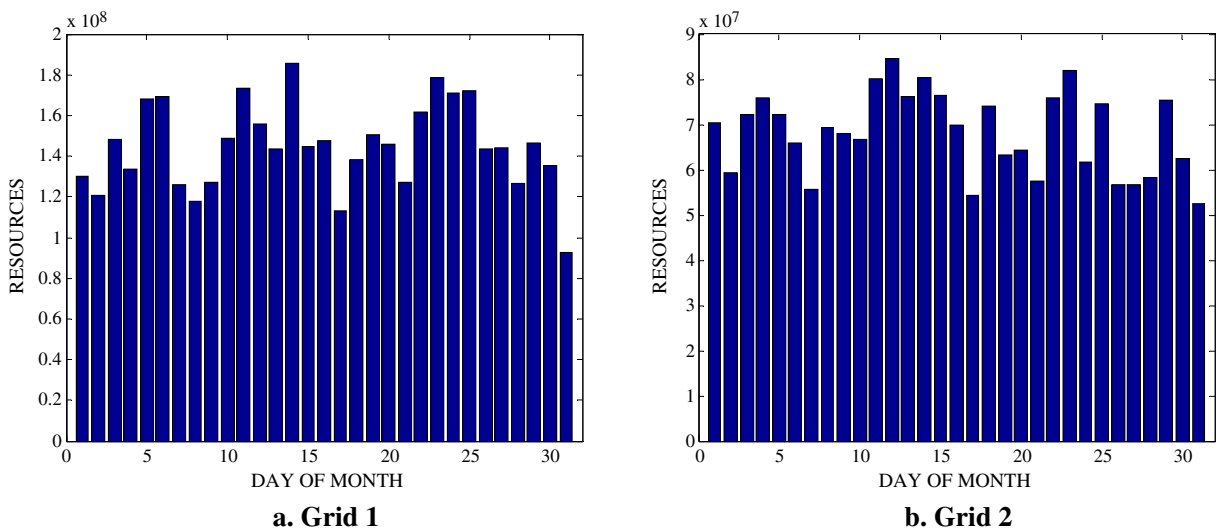
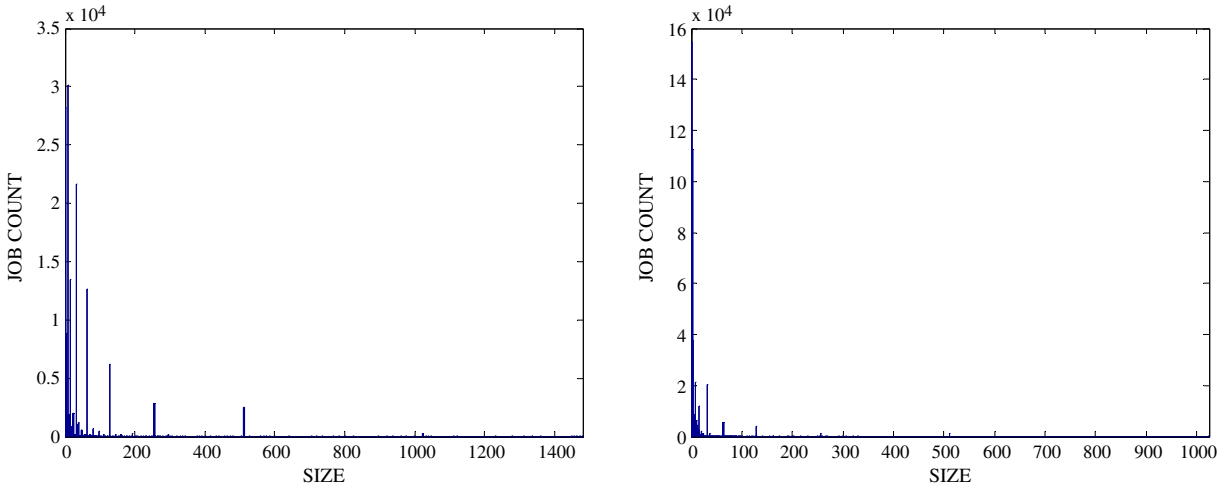
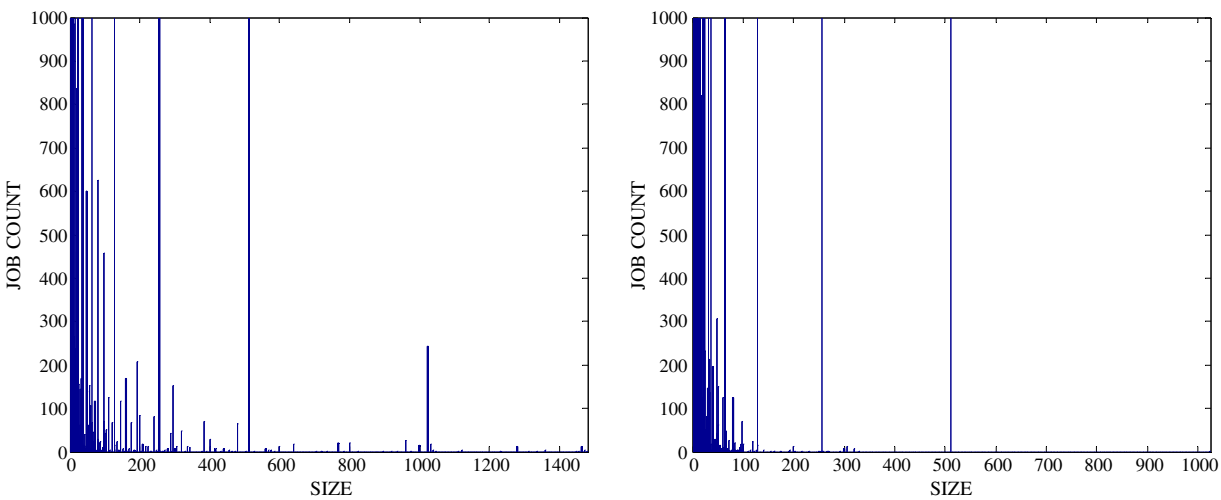


Fig. 10 Mean resource consumption in simulated period



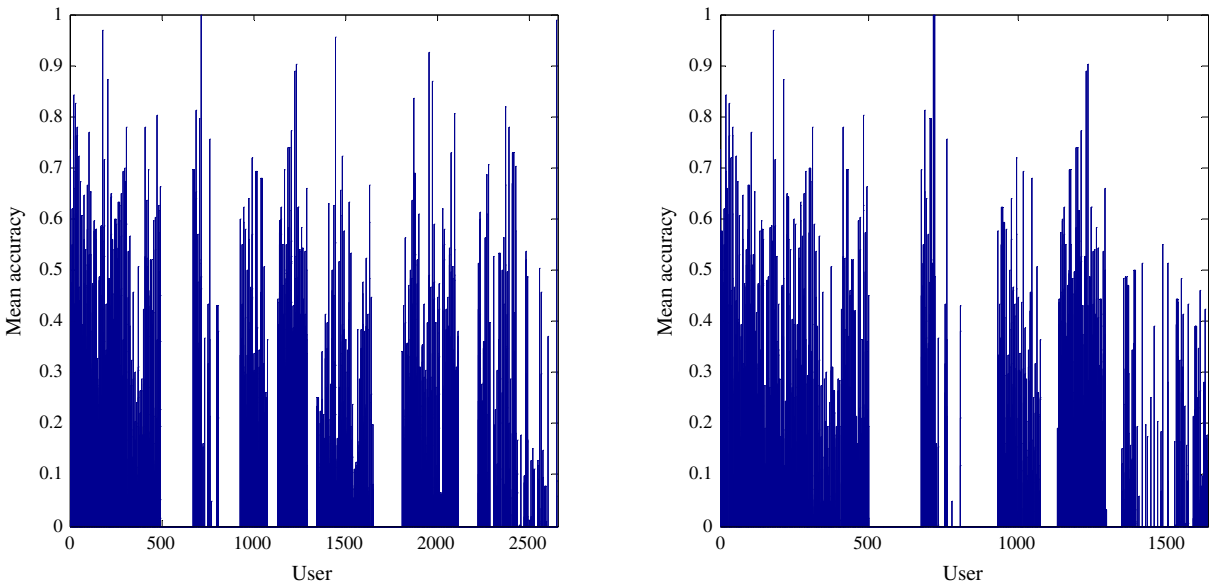
a. Grid 1 **b. Grid 2**

Fig. 11 Number of jobs per size



a. Grid 1 **b. Grid 2**

Fig. 12 Number of jobs per size (view is limited by 1,000 jobs)



a. Grid 1 **b. Grid 2**

Fig. 13 Accuracy histograms of user runtime estimates

Table 6 Percentages of the performance degradations for all test cases

Metric	Strategy													
	MCT	MLB	LBal_S	LBal_T	LBal_W	MLp	MPL	Random	MST	MSWCT_W	MWWT_S	MWT	MWWT_T	MWWT_W
ρ	0	0	0	16	1	1	0	12	0	14	4	0	1	6
SD	3618	750	15	52769	2743	77	0	35715	105	37224	17633	617	22968	100564
SD_b	2643	581	17	39005	2122	62	0	27060	101	25839	10063	416	17510	40650
Th	0	0	0	12	1	1	0	10	0	11	3	0	1	5
TA	70	26	1	2564	74	4	0	1134	4	1363	408	14	445	783
U	0	0	0	12	1	1	0	10	0	11	3	0	1	5
t_w	3045	1131	40	70264	3374	136	0	34978	166	38374	13217	593	19447	33637
$LBal_s$	2112	2159	195	1913	2597	416	118	1483	20	1186	166	50	2214	1769
$LBal_t$	612	1516	2033	989	1827	0	2046	1926	1933	4390	2094	532	411	1815
$LBal_w$	134	72	16	1066	30	231	82	801	0	671	165	44	204	224
SCT	0	0	0	2	0	0	0	1	0	1	0	0	0	0
$SWCT_s$	0	0	0	8	0	0	0	1	0	1	0	0	1	1
$SWCT_t$	0	0	0	4	0	0	0	1	0	3	1	0	0	0
$SWCT_w$	0	0	0	14	0	0	0	2	0	1	1	0	1	1
SWT	3045	1131	40	70264	3375	136	0	34978	166	38374	13217	593	19447	33638
$SWWT_s$	539	285	5	37910	700	115	3	5931	20	2745	2420	209	3550	3803
$SWWT_t$	1118	1341	29	108146	3433	249	0	38381	61	66011	15620	416	9855	20091
$SWWT_w$	621	164	8	102965	438	134	2	12331	35	8169	4929	283	4479	4964
WTA_s	117	62	1	9981	148	27	1	1485	4	696	590	45	787	879
WTA_t	4	4	0	796	10	1	0	234	0	474	95	2	33	79
WTA_w	22	6	0	3838	15	5	0	453	1	302	179	10	156	176
WWT_s	539	285	5	37910	700	115	3	5931	20	2745	2420	209	3550	3803
WWT_t	1118	1341	29	108146	3433	249	0	38381	61	66011	15620	416	9855	20091
WWT_w	621	164	8	102965	438	134	2	12331	35	8169	4929	283	4479	4964
Average	832	459	102	31315	1061	87	94	10565	114	12616	4324	197	4975	11331

References

1. Krauter, K., Buyya, R., Maheswaran, M.: A taxonomy and survey of Grid resource management systems for distributed computing. International Journal of Software: Practice and Experience (SPE) **32**, 135–164 (2002)
2. Rodero, I., Corbalan, J., Badía, R.M., Labarta, J.: eNANOS Grid resource broker. In: Advances in Grid Computing. European Grid Conference (EGC 2005), pp. 111–121. Springer, Amsterdam (2005)
3. Rodero, I., Guim, F., Corbalan, J., Goyeneche, A.: The Grid backfilling: a multi-site scheduling architecture with data mining prediction techniques. In: Talia, D., Yahyapour, R., Ziegler, W. (eds.) Grid Middleware and Services. Challenges and Solutions, vol. 8, pp. 137–152. Springer, New York (2008)
4. Elmroth, E., Tordsson, J.: An interoperable, standards-based Grid resource broker and job submission service. In: First International Conference on e-Science and Grid Computing, 2005, pp. 212–220. IEEE Computer Society, Melbourne, Vic. (2005)
5. Avellino, G., Beco, S., Cantalupo, B., Maraschini, A., Pacini, F., Terracina, A., Barale, S., Guarise, A., Werbrouck, A., Sezione Di Torino, Colling, D., Giacomini, F., Ronchieri, E., Gianelle, A., Peluso, R., Sgaravatto, M., Mezzadri, M., Prelz, F., Salconi, L.: The EU datagrid workload management system: towards the second major release. In: 2003 Conference for Computing in High Energy and Nuclear Physics. University of California, La Jolla, California, USA (2003)
6. Ranganathan, K., Foster, I.: Simulation studies of computation and data scheduling algorithms for data Grids. Journal Grid Computing **1**, 53–62 (2003)
7. Derbal, Y.: Entropic Grid scheduling. Journal Grid Computing **4**, 373–394 (2006)
8. de Lucchese, O.F., Huerta Yero, E., Sambatti, F., Henriques, M.: An adaptive scheduler for Grids. Journal Grid Computing **4**, 1–17 (2006)
9. Ernemann, C., Yahyapour, R.: Applying economic scheduling methods to Grid environments. In: Grid Resource Management: State of the Art and Future Trends, pp. 491–506. Kluwer, Dordrecht (2004)
10. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A.: On advantages of Grid computing for parallel job scheduling. In: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 39. IEEE Computer Society (2002)
11. Ernemann, C., Hamscher, V., Yahyapour, R.: Benefits of global Grid computing for job scheduling. In: Fifth IEEE/ACM International Workshop on Grid Computing (Grid '04), in Conjunction with SuperComputing 2004, pp. 374–379. IEEE Computer Society, Pittsburgh (2004)
12. Vázquez-Poletti, J.L., Huedo, E., Montero, R.S., Llorente, I.M.: A comparison between two Grid scheduling philosophies: EGEE WMS and Grid way. Multiagent and Grid System. Grid Computing, High Performance and Distributed Applications **3**, 429–439 (2007)
13. Schwiegelshohn, U., Yahyapour, R.: Attributes for communication between Grid scheduling instances. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid Resource Management: State of the Art and Future Trends, pp. 41–52. Kluwer, Norwell (2004)
14. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: A multicriteria approach to two-level hierarchy scheduling in Grids. J. Sched. **11**, 371–379 (2008)
15. Zikos, S., Karatza, H.D.: Resource allocation strategies in a 2-level hierarchical Grid system. In: Simulation Symposium, 2008. ANSS 2008. 41st Annual, pp. 157–164. Ottawa, Ont. (2008)
16. Chunlin, L., Layuan, L.: Multi-level scheduling for global optimization in Grid computing. Comput. Electr. Eng. **34**, 202–221 (2008)
17. Wäldrich, O., Wieder, P., Ziegler, W.: A meta-scheduling service for co-allocating arbitrary types of resources. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Wasniewski, J. (eds.) Parallel Processing and Applied Mathematics, vol. 3911, pp. 782–791. Springer, Heidelberg (2006)
18. Tchernykh, A., Ramírez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., Zhuk, S.: Two level job-scheduling strategies for a computational Grid. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Wasniewski, J. (eds.) 6th International Conference on Parallel Processing and Applied Mathematics PPAM 2005, LNCS, vol. 3911, pp. 774–781. Springer, Heidelberg (2006)
19. Zhuk, S., Chernykh, A., Avetisyan, A., Gaissaryan, S., Grushin, D., Kuzjurin, N., Pospelov, A., Shokurov, A.: Comparison of scheduling heuristics for Grid resource broker. In: Third International IEEE Conference on Parallel Computing Systems (PCS 2004), pp. 388–392. IEEE, Colima, Colima, México (2004)
20. Pugliese, A., Talia, D., Yahyapour, R.: Modeling and supporting Grid scheduling. Journal of Grid Computing **6**, 195–213 (2008)
21. Schwiegelshohn, U.: An owner-centric metric for the evaluation of online job schedules. In: Multidisciplinary International Conference on Scheduling. Theory and Applications (MISTA 2009), pp. 557–569. Dublin, Ireland (2009)
22. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer, P.L., Johnson, E.L., Korte, B.H. (eds.) Annals of Discrete Mathematics 5. Discrete Optimization II, pp. 287–326. North-Holland, Amsterdam (1979)
23. Naroska, E., Schwiegelshohn, U.: On an on-line scheduling problem for parallel jobs. Inf. Process. Lett. **81**, 297–304 (2002)
24. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in Grids. In: IEEE International Symposium on Parallel and Distributed Processing 2008 (IPDPS 2008), pp. 1–10. Miami, FL, USA (2008)

25. Garey, M.R., Graham, R.L.: Bounds for multiprocessor scheduling with resource constraints. SIAM J. Comput. **4**, 187–200 (1975)
26. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: Online hierarchical job scheduling on Grids. In: Priol, T., Vanneschi, M. (eds.) From Grids to Service and Pervasive Computing, pp. 77–91. Springer, New York (2008)
27. Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N.: Online hierarchical job scheduling on Grids with admissible allocation. J. Sched. **13**, 545–552 (2010). doi:10.1007/s10951-010-0169-x
28. Bar-Noy, A., Freund, A.: On-line load balancing in a hierarchical server topology. SIAM J. Comput. **31**, 527–549 (2001)
29. Pascual, F., Rzadca, K., Trystram, D.: Cooperation in multi-organization scheduling. Concurr. Comput.: Practice and Experience **21**, 905–921 (2009)
30. Zhuk, S.: Approximate algorithms to pack rectangles into several strips. Discrete Math. Appl. **16**, 73–85 (2007)
31. Bougeret, M., Dutot, P.-F., Jansen, K., Otte, C., Trystram, D.: A fast 5/2 approximation algorithm for hierarchical scheduling. In: 16th International European Conference on Parallel and Distributed Computing, Euro-Par 2010. Ischia, Italy (2010)
32. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Modeling user runtime estimates. In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) 11th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2005). LNCS, vol. 3834, pp. 1–35. Springer, Cambridge (2006)
33. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**, 789–803 (2007)
34. Chiang, S.-H., Arpaci-Dusseau, A.C., Vernon, M.K.: The impact of more accurate requested runtimes on production job scheduling performance. In: 8th International Workshop on Job Scheduling Strategies for Parallel Processing, pp. 103–127. Springer Verlang (2002)
35. Bailey Lee, C., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In: Feitelson, D.G., Frachtenberg, E., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. Springer, New York (2004)
36. 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)
37. Guim, F., Corbalan, J., Labarta, J.: Prediction of based models for evaluating backfilling scheduling policies. In: Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 9–17. IEEE Computer Society (2007)
38. Goyeneche, A., Guim, F., Rodero, I., Terstyanszky, G., Corbalan, J.: Extracting performance hints for Grid Users using data mining techniques: a case study in the NGS. The Mediterranean Journal of Computers and Networks (MEDJCN). SPECIAL ISSUE on Data Mining Applications on Supercomputing and Grid Environments **3**(2), 52–61 (2007)
39. Smith, W.: Improving resource selection and scheduling using predictions. In: Nabrzyski, J., Schopf, J.M., Weglarz, J. (eds.) Grid resource management: state of the art and future trends, pp. 237–253. Kluwer, Dordrecht (2004)
40. Tsafrir, D., Feitelson, D.G.: The dynamics of backfilling: solving the mystery of why increased inaccuracy may help. In: IEEE International Symposium on Workload Characterization (IISWC 2006), pp. 131–141. IEEE, San Jose, California (2006)
41. Zotkin, D., Keleher, P.J.: Job-length estimation and performance in backfilling schedulers. In: Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8 '99), pp. 39–46. IEEE Computer Society (1999)
42. Talby, D., Tsafrir, D., Goldberg, Z., Feitelson, D.G.: Session-based, estimation-less, and information-less runtime prediction algorithms for parallel and Grid job scheduling. Technical report, School of Computer Science and Engineering, Hebrew University of Jerusalem (2006)
43. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/
44. Grid Workloads Archive, TU Delft. http://gwa.ewi.tudelft.nl