J Grid Computing
DOI 10.1007/s10723-013-9274-3

On Efficiency of Multi-job Grid Allocation Based on Statistical Trace Data

Gábor Bacsó · Ádám Visegrádi · Attila Kertesz · Zsolt Németh

Received: 20 December 2012 / Accepted: 6 September 2013

Abstract The ever growing number of computation-intensive applications calls for utilizing large-scale, potentially interoperable distributed infrastructures. Nowadays, such distributed systems enable the management of heterogeneous scientific workflows of considerable sizes, where job scheduling and resource management is a crucial issue. In this paper we focus on the challenges of scheduling parameter sweep applications, a specific and commonly used type of workflows where ordering of job executions is irrelevant. A parameter sweep has a large set of independent job instances, called a multi-job, submitted for execution in a single step. In order to cope with the high uncertainty and unpredictable load of resources, and the simultaneous submissions of multi-job instances, we propose a statistics-based brokering approach for allocating jobs to resources so that the makespan is minimised. Earlier studies claim that users' predictions on job runtime are inaccurate and unusable for scheduling. Our aim is to examine, whether statistical trace data for the same purpose is efficient compared to randomized allocation.

Keywords Grid computing · Grid scheduling · Allocation · Grid brokering · Workload traces

1 Introduction

Researchers of various disciplines ranging from life sciences and astronomy to computational chemistry, create and use scientific applications producing large amount of complex data relying heavily on compute-intensive modeling, simulation and analysis. The ever growing number of such computation-intensive applications calls for the interoperability of distributed infrastructures including private and public clouds, Grids and clusters, generally modeled as Distributed Computing Infrastructures (DCI). Scientific workflows have become a key paradigm for managing complex tasks and have emerged as a unifying mechanism for handling scientific data. Workflow applications capture the essence of the logic of scientific process, providing means to describe

G. Bacsó · Á. Visegrádi (✉) · A. Kertesz · Z. Németh
MTA SZTAKI, P.O. Box 63, 1518 Budapest, Hungary
e-mail: visegradi.adam@sztaki.mta.hu

G. Bacsó
e-mail: bacso.gabor@sztaki.mta.hu

A. Kertesz
e-mail: kertesz.attila@sztaki.mta.hu

Z. Németh
e-mail: nemeth.zsolt@sztaki.mta.hu

Published online: 28 September 2013

abstractly as data- or control-flows. During the execution of a workflow, its jobs are mapped onto resources of DCIs, to perform large-scale experiments.

The combination of heterogeneous scientific workflows, and their execution in a large-scale system consisting of multiple DCIs, including academic and public Clouds were targeted by the European SHIWA project [23] and its successor, ER-flow [24]. A large number of workflows belong to the "parameter study" or "parameter sweep" class where a large input range is explored by applying algorithms on each item of the input set independently [1, 2] hence, they are also called multi-job applications. In this work we consider this practically very important subset of workflows and we entirely focus on multi-jobs.

Parameter study jobs or multi-jobs are typically executed by submitting all instances simultaneously to the scheduling component, also called broker, of the workflow management system that makes the necessary assignments of resources to job instances. Note that the temporal ordering of execution is irrelevant in case of multi-jobs hence, the mapping problem that comprises of scheduling (when to execute) and matching (where to execute) is reduced to simple matching for the sake of unambiguity we will call this aspect as *allocation* in this paper. The broker carries out the allocation task so that the distribution of the job instances approximates optimum according to some criteria that may be execution time, cost, power consumption, quality requirements, resource usage just to mention a few. We consider the most common criterion, execution time more precisely, makespan of the multi-job. Thus, the broker delivers an allocation so that the entire set of job instances finishes as early as possible. More details and definitions of the infrastructure and the allocation model are in Section 3.

This goal has been aimed at in many works, see Section 2, and many sorts of allocation strategies, algorithms and heuristics have been proposed. A very common issue of all these approaches is the lack or the imprecision of the (estimated) data on task execution times. Earlier studies claim that job runtime predictions of users are inaccurate and unusable for Grid scheduling [16]. In order to cope with the high uncertainty and unpredictable load of these infrastructures and with the simultaneous submissions of multi-job instances, we propose a novel alternative, statistics-based brokering approach. We base our allocation algorithm on historical data on execution times and apply probabilistic calculations. Our aim is to examine, how much statistical trace data for predicting job runtimes can improve some allocation methods compared to baseline experiments. The proposed methods are experimented by simulations realized in GridSim and PythonSim.

Our main contributions are: (i) the design of multi-job allocation algorithms managing resources of multiple DCIs based on statistical trace data, (ii) development of a proper simulation environment that enables the examination of these algorithms, and (iii) the evaluation of our proposed approach using simulations.

The remainder of the paper is as follows: Section 2 presents the related resource management approaches; Section 3 describes our considered model for Grid Allocation; Section 4 introduces the allocation algorithms based on statistical approaches. Finally, Section 5 discusses the performed evaluations, and the contributions are summarized in Section 6.

2 Related Work

Our research is partly motivated by Lee et al. in [16], who investigated the accuracy of users' runtime estimates in batch scheduling systems. Typically users provide such estimates in time-limited infrastructures, i.e. their job is terminated after a given amount of time. Hence, users may overlook the potential benefits of accurate estimates, which is an essential requirement for a good schedule. The study concludes that even if a tangible reward is granted for accuracy, there is no substantial improvement in the overall average accuracy. Our aim is to investigate the reliability of runtime estimations calculated from trace files of categorized jobs in real Grid systems.

Ramirez-Alcaraz et al. [19] have analyzed different Grid allocation strategies depending on the type and amount of information they require, and they found that information about users' runtime estimate and local schedules does not

help to significantly improve the outcome of the allocation strategies. They concluded that quite simple schedulers with minimal information requirements can provide good performance. Practice seems to adapt to these findings, because too complex, sophisticated scheduling algorithms are rarely used in Grid brokers.

Hirales-Carbajal et al. [11] present an experimental study of 22 deterministic non-preemptive multiple workflow scheduling strategies in Grids. While their objective is to schedule and execute the whole workflow, and minimize its makespan, we restrict ourselves to parameter study jobs of such workflows.

Nevertheless, GridBot [25] represents an approach for execution of bags-of-tasks on multiple Grids, clusters, and volunteer computing Grids. It has a Workload Manager component that is responsible for brokering among these environments, which is similar to our approach, but they focus on tasks more suitable for volunteer Grids.

Oprescu et al. [20] propose a budget constraint-based resource selection approach for Cloud applications. In this work they present a budget-constrained scheduler called BaTS, which can schedule bags of tasks onto multiple clouds with different CPU performance and cost, minimizing completion time with maximized budget. Their scheduler learns to estimate task completion times at run time, while we use estimations by statistical calculations from real world trace files.

Cirne et al. [5] developed Workqueue with replication (WQR) that keeps computing resources busy so that if a node runs out of work and would become idle, a copy of an unfinished task is assigned to it. In such a way a second chance is given for a faster execution: the two copies of the same task compete. The approach does not need any information about the tasks or the resources and can dampen the effect of dynamic loads and improve execution time by wasting some (controlled limit) resources. Following this idea Da Silva et al. in [6] also raise the question of methods that require information compared to WQR. WQR is however, iterative as opposed to our single decision making for allocation.

Casanova et al. [4] focused on the very same problem of scheduling parameter sweep applications on Grids, with particular attention to file transfers and network performance. Also, their motivation is in alignment with most of the papers in this area: inaccurate predictions can largely mislead scheduling. Their approach is modifying existing heuristics so that they are adaptive in a dynamic heterogeneous environment. The core of the scheduling is a Gantt chart that is created and updated periodically and keeps track of job and resource assignments. Assignments are governed by a heuristics called *sufferage* where a task is assigned to a host if the task would "suffer" the most if done otherwise; "suffering" is expressed as the difference between the best and the second best minimum completion times. A considerable complexity is added in this model by taking into account file transfer and communication costs; this lead to modifying the definition of the sufferage algorithm.

Maheswaran et al. [17] addresses the issue of mapping independent tasks onto heterogeneous computing systems. They apply heuristics aiming at optimizing for *throughput*, i.e. increasing the finished task per time unit ratio. It considers Minimum Completion Time (MCT), Minimum Execution Time (MET), Switching Algorithm (switches between MCT and MET) and k-percent Best (MCT on a subset of resources) as on-line heuristics whereas Min-Min, Max-Min and Sufferage for batch mode ones. Min-Min and Max-Min roughly correspond to MCT and MET of on-line algorithms. Their analysis revealed that batch scheduling can outperform on-line scheduling for large number of tasks on the other hand some on-line scheduling may have lower computation time. Within the batch scheduling class, Min-Min and Sufferage were proven to be superior.

Menascé at al. [21] introduce further static scheduling by combining an envelope (a selection of tasks and resources to be considered) and a heuristics. In the paper they derived 6 static scheduling methods by crossing three task selection and two processor selection heuristics; these were compared with three dynamic ones. They claim the superiority of static methods: these are executed rarely hence, more sophisticated algorithms can be realized without time penalty. All these methods are based on task priorities whereas, in our case we do not differentiate between task priorities.

3 Our Model for Grid Allocation

The notion of historical data We are seeking a mapping solution for assigning jobs of "parameter study" type to resources so that the assignment minimizes the makespan, i.e. minimizes the time that takes the execution of the job that finishes last. Job scheduling on a multiprocessor system has been studied for more than 30 years and is known to be *NP-hard* [26]. Scheduling in Grid systems, become more complicated with multi-organizational shared resources, therefore Grid scheduling is also NP-hard [8, 22]. Solutions thus, apply some sort of heuristics and approximations as listed in Section 2.

Some of these methods are based on runtime estimates and the inaccuracy of these estimates is a perennial problem mentioned in the job scheduling literature. Even if users are required to provide these values, there is no substantial improvement in the overall average accuracy [16]. Our approach tries to find alternative solutions by gathering data from trace files and establishing allocation on statistical properties. The basis for this statistical approach are workload traces published in the Parallel and Grid Workload Archives [10, 18]. Some of the traces contain categorization for jobs based on *group, user* and *execution* identifiers, therefore we were able to perform statistical analysis on refined and better structured trace data and apply it to allocation algorithms.

The notion of resources Our approach optimizes resource utilization of Grid systems consisting of a given number of resources. We suppose that there is a trace file available for the actual Grid system we consider, which contains historical job execution data. We use statistical information from these data to predict load on Grid resources. We model resources with the notion of Distributed Computing Infrastructure (DCI). This is a hypothetical resource that can be realised either as a cluster, Grid, desktop Grid or other types of distributed computing system while provides a unified and standard interface. In this way we can model the infrastructure in an abstract way without focusing on its actual physical realisation. Hence, in this paper we call unit of computing infrastructure as "resource" but in reality it can be a single processor (computer) or a set of processors (a cluster). With respect to scheduling these resources are the smallest units that can be observed or controlled. Obviously, resources may contain smaller (finer-grained) computing elements like cores of a processor or processors of a cluster. Albeit some models will take into account them, these elements are managed by an operating system or a middleware and are assumed hidden from the scheduling or mapping agent. Note, that in this way our solution is implementation independent: if the necessary information (traces, historical data, etc.) on other DCIs (e.g. clusters or clouds) are provided, our proposed approach could be applied in those cases, too.

The notion of tasks In this paper we focus on "multi-job", a large set of tasks to be executed; they logically form a single program but otherwise they are completely independent, neither communication nor synchronisation is between tasks. This class of jobs forms a special case of workflows where a broad fan of parallel branches are executed in parallel. Typically they are "parameter studies" or "parameter sweep" applications running the same algorithm on different input parameters. A similar notion of independent tasks appear as bag-of-tasks (BoT) in the general parallel programming literature but they slightly differ from parameter sweep applications as (i) tasks can be of arbitrary type as long as they are independent hence, their sizes cannot be assumed similar. Furthermore, we also assume that (ii) the tasks to be executed are available at the same moment, they are mapped in a batch-like fashion at a single mapping event and task arrival times are not applicable. Parameter sweep or parameter study applications are a subset of BoT type.

The notion of the resource broker Executing a user application in a Grid environment requires several prerequisites. Users need to learn the interfaces of the Grid services and need to describe their application prior to submissions. Production Grid systems may consist of hundreds of thousands of resources (e.g. 240,000 processor cores in EGI [7]), therefore it is not realistic to assess the actual state of computing and storage resources and select some for an application but there is a

clear requirement for automated resource discovery. Special resource managers, also called as resource brokers are meant to solve this problem or, in a more general sense the problem of scheduling. As resource management is a key part of current Grid middleware solutions, most middleware developer groups and projects have developed their own tools for resource brokering.

4 Proposed Algorithms

In our examinations we consider the following simplified Grid model: we have a heterogeneous Grid system consisting of resources having some workload simulated by real world traces. The goal of our algorithms is to map jobs optimally to these resources at a certain time of submission. These jobs represent a multi-job of a parameter study workflow, and they execute the same algorithm over different input parameters. The quantities used in the proposed algorithms are summarized in Table 1.

4.1 Deterministic Case

Though the deterministic approach we discuss in this subsection is a very simplified version of reality, it is a good method (just from an algorithmic point of view), to solve and use it for the non-deterministic cases. Nevertheless, this is possible, since we calculate with the expected values of the random variables only.

The aim of Algorithm 1 is to distribute k jobs (representing multi-jobs of a parameter study workflow) on n resources (representing the nodes of a heterogeneous Grid) so that resource i has a certain pre-load c_i – the number of jobs assigned to resource i already belonging to different applications. Further jobs, running on resource i are "our jobs", k_i is their number in a hypothetical allocation, see Fig. 1. It is assumed that both the jobs to be distributed and the other jobs have *unit* execution times thus, k_i jobs take k_i time and c_i is equivalent to a delay of c_i time units. Furthermore, pre-loads c_i are ordered in ascending order: $c_1 \leq c_2 \leq c_3 ... \leq c_n$. A certain node finishes its operation after $R_i = c_i + k_i$ time units. We define turnaround time $T(\mathbf{x}) = \max_{i,k_i \neq 0}(R_i)$, the maximum of the execution times for a certain job distribution. The goal is to minimize T, i.e. find a distribution $\mathbf{x}$ so that $T = \min_{\mathbf{x}} T(\mathbf{x})$. Algorithm 1 can be applied for any non-negative real numbers c_i and provides an optimum solution in this case.

Line 2 of Algorithm 1 initializes the result vector x, then in line 3 the index of the resource with the highest load (n) is stored in u which will

Table 1 Summary of quantities used in the algorithms

$n \in \mathbb{Z}$	Number of all available nodes
$k \in \mathbb{Z}$	Number of jobs to be allocated
$k_i \in \mathbb{Z}$	Number of allocated jobs on node i
$t_j \in \mathbb{R}$	Execution time of a single job j
$c_i \in \mathbb{R}$	Delay (waiting) caused by other jobs (pre-load) on node i
$\mathbf{x} = (k_1, k_2, ...k_n)$	Vector of number of allocated jobs on node i
J	Expected value of t_j, $\mathbb{E}(t_j)$
R_i	Running time (pre-load + own jobs) on node i
T	Makespan (total execution time)

Algorithm 1 Unit execution times

Data: $k, n, (c_1, c_2 ... c_n)$
Result: $\mathbf{x}$

1 **begin**
2 $\quad \mathbf{x} \leftarrow [0, 0, ...0]$
3 $\quad u \leftarrow n$
4 $\quad$ **while** $\sum_{i=1}^{u-1} (c_u - c_i) \geq k$ **do**
5 $\quad\quad u \leftarrow u - 1$
6 $\quad$ **end**
7 $\quad$ **forall the** $i \leq u - 1$ **do**
8 $\quad\quad k_i \leftarrow \lfloor c_u - c_i \rfloor$
9 $\quad$ **end**
10 $\quad JobsBelow = \sum_{i=1}^{u-1} k_i$
11 $\quad JobsAbove = k - JobsBelow$
12 $\quad Layers = \lceil JobsAbove/u \rceil$
13 $\quad Leftovers = k \mod u$
14 $\quad$ **forall the** $i \leq u$ **do**
15 $\quad\quad k_i \leftarrow k_i + Layers$
16 $\quad$ **end**
17 $\quad$ **forall the** $i \leq Leftovers$ **do**
18 $\quad\quad k_i \leftarrow k_i + 1$
19 $\quad$ **end**
20 $\quad T \leftarrow \max_i k_i$
21 **end**

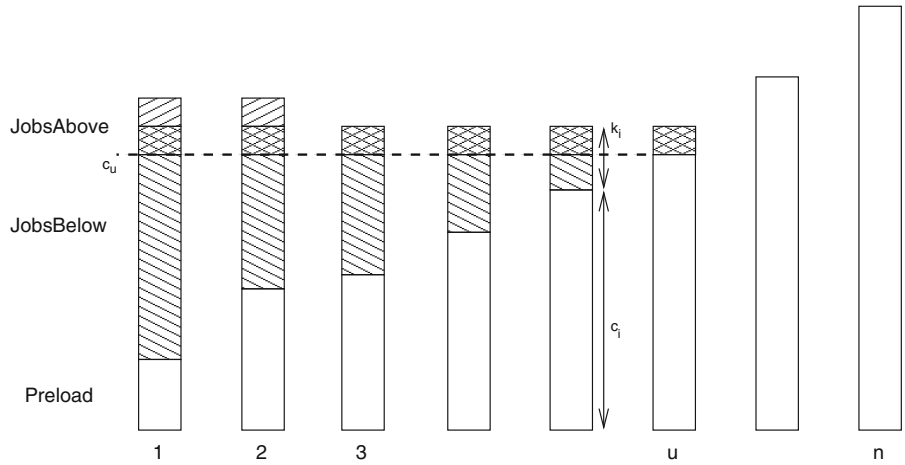


Fig. 1 Pre-loads and job distribution on n resources

keep track the number of required resources, see Fig. 1. Line 4 checks if all jobs fit on $u-1$ resource so that the load levels are below c_u, if so, u is decremented and checked again. At the end of loop, line 8, jobs are distributed on $u-1$ resources so that the level of load on each resource would be c_u. The assigned jobs add up $JobsBelow$ in line 11. The remaining jobs, not assigned to any resource so far are $JobsAbove$ calculated in line 12. These form *Layers* layers on top of c_u load on resources $1..u-1$ and an additional job on resources $1..Leftovers$; these are calculated in lines 13–20, see also Fig. 1.

4.2 Non-deterministic Case

In the following the execution times of jobs will not be uniform but will have a certain probabilistic distribution and represented by their expected value $J = \mathbb{E}(t_j)$ for all j. We can use this method only when t_j can be considered as random variables with the same distribution. This is the case when the statistical dispersion is small. Thus, we obtain a good estimation of the real process. Delay times on node i are γ_i. Hence, the total execution time on node i is the sum of the expected value of delay and the expected value of k_i jobs,

$$R_i = \mathbb{E}(\gamma_i) + k_i * J$$

Dividing by J we obtain

$$\frac{R_i}{J} = k_i + c_i$$

where $c_i = \frac{\mathbb{E}(\gamma_i)}{J}$. This is the reason why we emphasize that c_is are not restricted to integers in Algorithm 1. Hence, Algorithm 1 is applicable with these substitutions if standard deviation is small. The turnaround time can be obtained as the calculated turnaround multiplied by J.

Algorithm 2 Non-deterministic execution times

```
Data: k, n, (γ1, γ2...γn), J
Result: x
1 begin
2     forall the i ≤ n do
3         c_i ← E(γ_i)/J;                    // Calculate c_i
4     end
5     // The body of Algorithm 1
6     T ← T * J
7 end
```

5 Evaluation of the Proposed Algorithms

In order to evaluate the previously introduced algorithms, we propose four scenarios for experiments as summarized in Table 2.

The ultimate problem we planned to address, where the scheduler has no apriori knowledge about the background load of the managed resources, which case is known to be *NP-hard*. In our model it is represented by Scenario 4, in which we know an estimated run time for each workload jobs in the system, which is still an NP-hard case. In order to simplify this case, we reduce

Table 2 Experimental scenarios

Scenario	Execution environment	Pre-load jobs	Submitted jobs (to be allocated)
1	n resources, 1 node per resource and 1 processor per node	Uniform run time, arrival as in trace file	k jobs; uniform, deterministic run-time
2	<same as above>	Arrival and run time as in trace file	k jobs, run-time of independent identical probability distribution
3	<same as above>	Arrival and run time as in trace file. Jobs grouped and expected value of run time for each group is known	k jobs, with a) uniform, deterministic run-time b) jobs grouped and expected value of run time for each group is known
4	n resources, m node per resource and l processor per node	<same as above>	k jobs, uniform, deterministic run-time

the uncertainty by concretizing some parameters step-by-step. In Scenario 3 we restrict ourselves to examine Grids having resources with a single processor, and in Scenario 2 we consider only a single estimated run time for all workload jobs – both cases are still NP-hard. Then we arrive to Scenario 1, where the exact run times are known, and all these run times are the same. This last case is not NP-hard anymore, so we can give an optimal allocation for these simulations.

Having this in mind, our research approach in the evaluations is the following: we start with the easiest case, Scenario 1, and execute simulations up to Scenario 3, while we focus on the effectiveness of the algorithms in the cases of increasing complexity. The measured results present, how the effectiveness changes from certainly known run times to statistically estimated ones compared to the randomized resource selection.

Due to the largely different nature of Scenario 1 (exact solution is possible) and the other scenarios, we created two different simulation environments to evaluate our proposed approach. The first one is a Python-based simulator (capable of handling artificially generated job data) that we used for examining Scenario 1 and Scenarios 2 – these experiments are simplified in comparison to real practical cases therefore, a simple lightweight simulator is sufficient. The second one is an extension of the well-known GridSim simulator [3] applied to real-world experiments, which we extended to cover Scenario 1, 3 and 4. The following subsections describe these simulators and the performed evaluations.

5.1 Simulations with PythonSim

We have created a specialized simulator that allowed us to test and verify our model and algorithms with artificially generated job data. This simulator was designed to be closer to our model than universal discrete event simulators; this makes the implementation simple, fast and reliable. Because of its speed, we were able to run simulations of the Scenario 1 and Scenario 2, described in Section 5, with many input parameters, in a short time.

5.1.1 Simulation Environment

PythonSim is a simple discrete event simulator with no support for message passing. It is implemented in Python; and we used the Just-in-Time compiler *pypy* to execute the simulations. Events have *timestamp* and *priority* attributes, and can be *executed*. Upon execution, events can store necessary information, and may create new events; for example, a job arrival event at t_a will create a job start event at $t_s = t_a + \text{waittime}$, which will in turn create a job completion event at $t_c = t_s + \text{runtime}$ that will, on execution, save its timestamp t_c as the latest job completion time.

These events are stored in a priority queue, in which they are ordered lexicographically on $(\text{timestamp}, \text{priority})$. The queue is initialized with known events, based on the simulation to be performed, and its parameters. Then, in iteration, the first event in the queue is executed and removed from the queue, until the queue becomes empty or an 'EndOfSimulation' event is

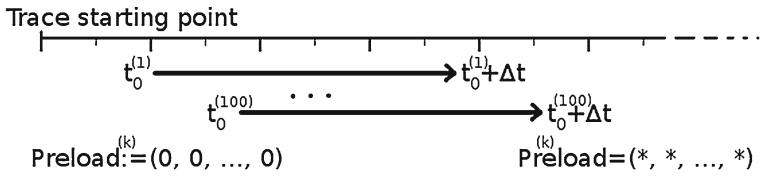


Fig. 2 Determining multiple pre-load vectors from a trace

executed. This simple, sequential implementation of the simulation environment makes it fast and free of race-conditions.

Simulation of allocation procedures were executed as a sequence of two "sub-simulations", both performed in this simulation environment. In the first sub-simulation, we re-executed the historical traces, generating pre-load vectors to be used as input for the algorithms. Then, in the second round, each pre-load vector was reused in several simulations, performing job allocation based on the pre-load and determining the resulting makespan according to specific simulation parameters.

5.1.2 Scope and Parameters

In these simulations, we focused on Scenario 1 and Scenario 2 described in Section 5. We assumed that the pre-load vectors generated were accurate and deterministic. We also assumed that, at each DCI, allocated jobs are executed in a FIFO manner, on a single core; which means that neither jobs arriving later nor already allocated jobs will affect the execution time of jobs currently being allocated. This is in contrast with our simulations based on GridSim, described later in Section 5.2, where we do not make these assumptions.

We have performed evaluations based on the SHARCNET historical trace; using both artificial *execution times* and the original execution times provided in the trace to create pre-load vectors. We determined 100 points randomly in time ($t_0^{(i)} \in$ [trace_start..trace_end], $i \in [1..100]$), from each point a simulation was started, we let the simulation run for Δt seconds where the simulation was interrupted, and we used the momentary state of the execution queues as a pre-load vector (Fig. 2).

These pre-load vectors were used as input for the job allocation algorithms. The number of jobs to be allocated (k), and the *distribution* of their execution times were the other parameters of the simulation. The parameters used are described in Table 3.

Table 3 Description of simulation parameters

Execution time $\in \{1000, 5000, 10{,}000,$ original from trace$\}$
t_0: Fixed 100 points in the trace, generated using *random.org*
Δt: 1×, 5×, 10× multiple of the execution time; and 8 days in the case when the original execution times were used
$k \in \{10, 100, 1000, 10{,}000\}$
(i.i.d.) Execution times of the k submitted jobs:
– Deterministic; 1000
– Gauss; $\mu = 1000,\ \sigma = 100$
– Transformed Pareto: $\tau \sim 500 * (\xi)$; where ξ has a Pareto distribution with scale= 1 and shape $\approx 2 + 2^{-12.288}$. $\Rightarrow E(\tau) \approx 1000$, and $D(\tau) \approx 100$. The Pareto distribution was chosen because it is generally a good model of the execution time of jobs.

5.1.3 Simulation Results

Figure 3 shows the distribution of load in the case where the original execution times were used.

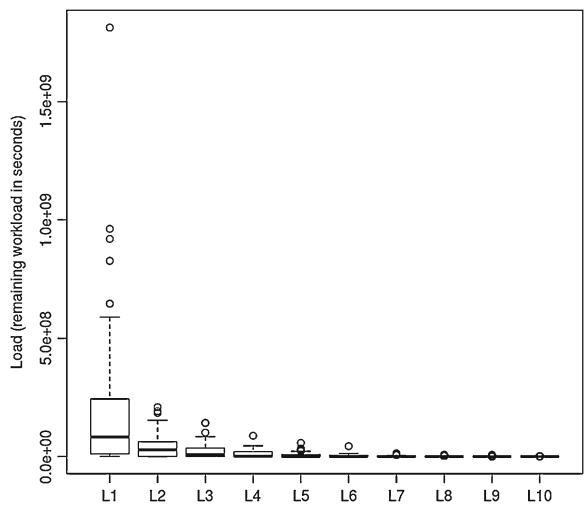


Fig. 3 Boxplot of the 100, individually ordered pre-load vectors

Each generated pre-load vector was individually ordered, as the algorithms treat all DCI-s equally. The results show that the all-time load on the system exhibits a Pareto-like distribution; that is, most of the DCIs in the system have little or no load, the current load in the system is always concentrated to a few DCIs.

Based on the generated pre-loads, we have conducted several experiments to evaluate the algorithms. For each pre-load vector, we have simulated the allocation and execution of k jobs using randomized allocation and Algorithm 1.

We have found that the results are hardly affected by distribution of the length of the allocated job (assuming i.i.d.). The results of the deterministic and the Gaussian distribution-based cases are indistinguishable; while the makespans in the Pareto distribution-based case are only slightly higher: the average makespan in the Gaussian or distributed case was < 0.1 % smaller than in the Pareto case. Figure 4 shows the average makespans and their linear regressions in the Pareto case. Each line and the corresponding data points represent the results of simulations based on a specific setting for pre-load job length.

There are four overlapping results on the bottom of this figure. In the first three cases (pre-load job length $\in \{1000, 5000, 10000\}$) the results of our algorithm almost line up, because the load on most DCIs was low, allowing the algorithm to produce low-makespan allocations. In the case where the pre-load job length is 1000, the *random* algorithm produced similar results to Algorithm 1, as in this case, pre-load vectors were uniform. If the pre-load is uniform, the random algorithm approximates the optimal allocation well.

However, as the pre-load jobs' length is increased, pre-load vectors start to resemble the boxplot in Fig. 3. That is, the vector becomes non-uniform, in which case the random algorithm cannot perform well. The two thin lines on Fig. 4 (case: random algorithm, pre-load job length $\in \{5000, 10000\}$) breaks away from the cluster as the pre-load vectors become skewed in these cases.

The thick, solid line separated from the others on Fig. 4 represents the result of the case, where the original run-times were used as pre-load job lengths. In this case, the pre-load vectors have the distribution shown on the boxplot, with high *minimal* loads, affecting the makespan of the jobs allocated by Algorithm 1. On the other hand, the *maximal* values are extremely high in this case, which affects this algorithm much less than the random algorithm. Using the random algorithm, the makespan is so high, it is not displayed on this figure—the two lines on Fig. 5a show the results and their relation in this case.

Figure 5 shows all results where the execution time of the allocated jobs exhibited a transformed Pareto distribution. Each symbol in the diagram shows the makespan of k jobs allocated based on a particular pre-load vector. (Each *visible* symbol in the diagram is actually the overlapping of many close results.) The circle marks are the makespans

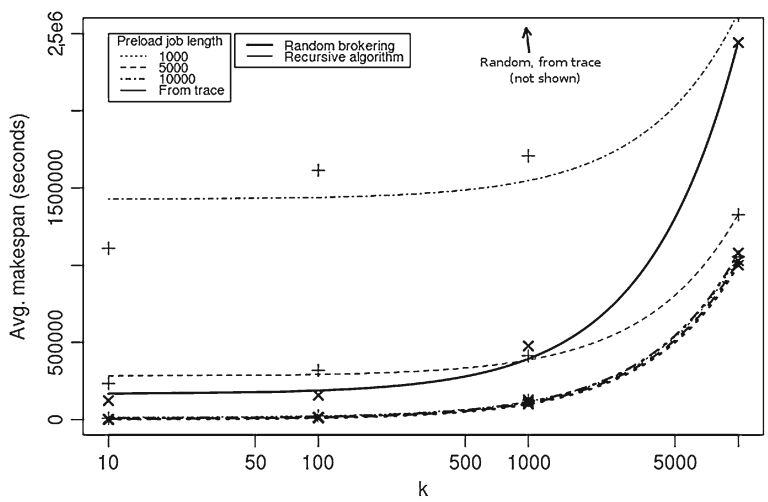
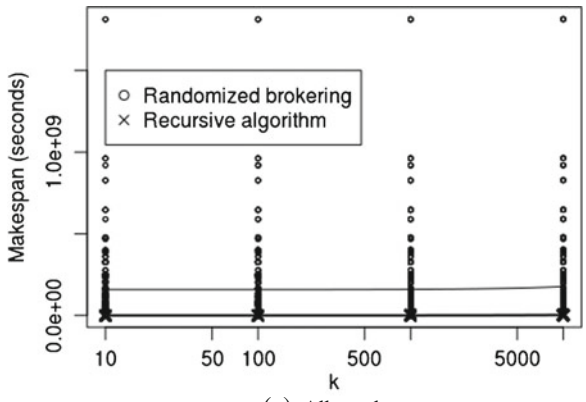
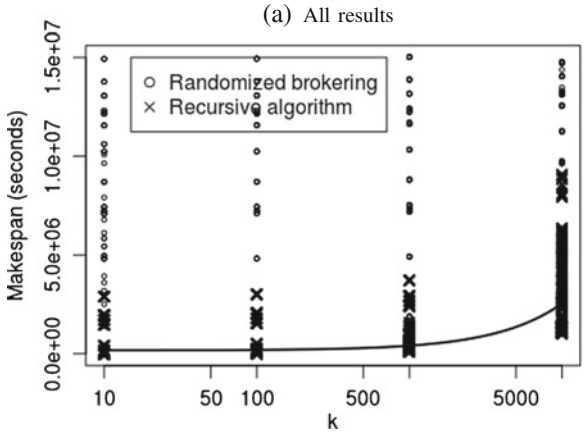


Fig. 4 Average makespans in different cases of the simulation



(a) All results



(b) Results where makespan < 1.5*e*7

Fig. 5 Simulational results where execution time $\sim \tau$ (see: Table 3)

generated by the random allocation method, while the cross marks are the result of Algorithm 1. Lines in these diagrams represent the linear regression model of the result sets; thick lines are used for Algorithm 1, and the lines for the random brokering case.

It is clear, that the results achieved with Algorithm 1 are far better than that of the random algorithm. The uniform results of the random algorithm is because the makespan in this case depends on the number of submitted jobs, the number of DCI-s, and only the maximal element of the pre-load vector; which, in our case, is higher than the load of the allocated jobs by orders of magnitudes. Note, that symbols lining up in a row are not exactly level, but linearly increasing, which can be seen with high zoom.

5.2 Simulations with GridSim

5.2.1 Simulation environment

For performing realistic and reproducible evaluations, we have chosen the GridSim Toolkit [3] to create a robust simulation environment. It supports modeling and simulation of heterogeneous Grid resources, users, applications, brokers and local schedulers in a Grid Computing environment. It provides primitives for the creation of jobs (called Gridlets), mapping of these jobs to resources, and their management, therefore resource brokers can be simulated to study allocation algorithms. It provides a multilayered design architecture based on SimJava [12], a general purpose discrete-event simulation package implemented in Java. All components in GridSim communicate with each other through message passing operations defined by SimJava.

Within GridSim, resources consist of one or more machines, to which workloads can be set. As an extension of GridSim classes, we have developed the 'GridSimStatQueueBroker', 'SimWorkload' and 'SimulatorSetup' entities in order to enable the simulation of the previously mentioned scenarios. On top of these simulated Grid infrastructures we can use the broker entities for setting up brokers with various allocation policies, the 'SimWorkload' entities are used for submitting the workload jobs, while the 'SimulatorSetup' component is responsible for parameterizing and executing each experiment.

The 'GridSimStatQueueBroker' is an extended 'GridUser' entity:

- it can be connected to one or more resources;
- various allocation policies can be defined (predefined ones: *stat* – resource selection based on statistics, *rnd* – random resource selection, *fcpu* – resources having more free CPUs or less waiting jobs are selected, *nfailed* – resources having less machine failures are selected);
- it has a queue with length k to store the submitted job instances;
- finally it stores the estimated waiting time for the queues of each resource to a local database

(which can be based on real or statistically estimated run times).

The 'SimWorkload' entities are extended 'GridSim' entities:

- they are used for submitting the workload jobs from the trace files to the machines;
- they also report the submissions and the group types of the submitted jobs to the broker, to enable updates of waiting queue length for the managed resources.

The 'SimulatorSetup' is an extended 'GridSim' entity:

- it can generate a requested number of gridlets (jobs) with different run time (length) or group identity;
- it is connected to the created broker and is able to submit the generated jobs to it;
- finally, once the jobs are finished, it generates a summary of them listing the simulated execution times.

We have also developed a scripting layer for the simulator in order to automatize and easily parametrize the simulation runs, and to generate plots.

Table 4 shows the parameters of the performed evaluation runs.

In order to address universality, we use real user application execution traces as background workload gathered both in parallel and production Grid environments, published in available archives [10, 18]. We have selected two workload trace files from the Grid Workloads Archive (GWA) [13]: (i) the GWA-T-10 SHARCNET file, which contains accounting records up to a year from the SHARCNET clusters installed at several academic institutions in Ontario, Canada, and (ii) the GWA-T-1 DAS2 trace file containing records for almost two years provided by the Advanced School for Computing and Imaging, the owner of the DAS-2 system.

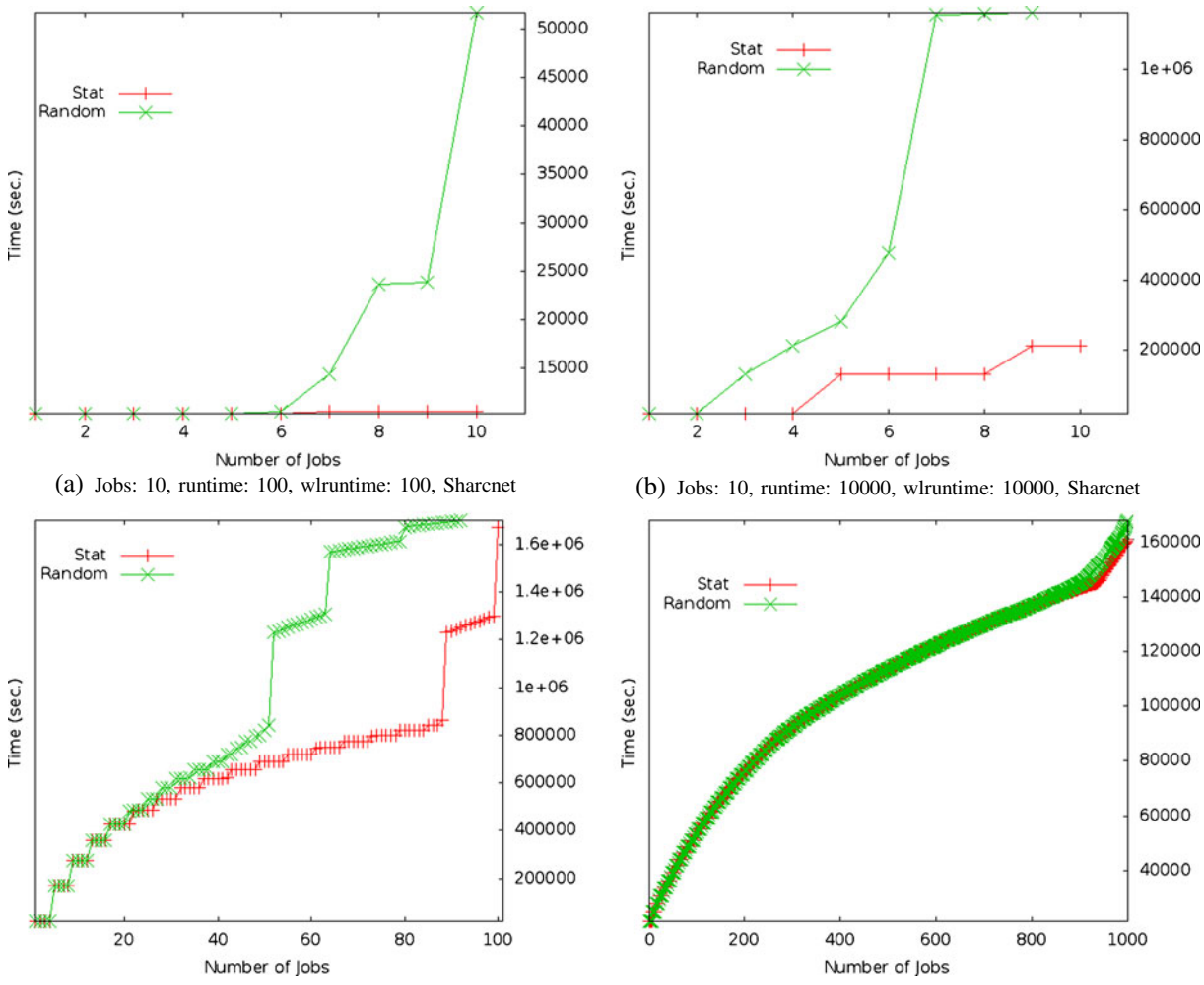
The main reasons for choosing these traces were that they contain the largest number of jobs, and include group identities for the logged jobs: the SHARCNET denotes the *Execution id* of the jobs, while the DAS2 denotes the *Group, User* and *Execution ids* of all jobs. These categories mean that the considered job belongs to the same group of users, or to the same user or an instance of the same application, respectively. These traces have been partitioned according to

Table 4 Evaluation parameters for Scenarios 1, 3 and 4

Scen.	Jobs	Job run time	Pre-load arrival and run time	Start time	Delay time
1	10	100	SHNET: 100	18607865	10000
		1000	SHNET:1000		
	100	5000	SHNET:5000	23077573	
	1000	10000	SHNET:10000	28937559	
3/a	100	100	as in DAS2 traces	100	1000
				868036	
	1000	500		6909476	
		500		12274286	69120
	5000	1000		23077573	
3/b	100	DAS2: EXE	as in DAS2 traces	100	1000
				6909476	
	1000	DAS2: USER		12274286	
				23077573	
	5000	DAS2: GROUP		28937559	
	10000			33331397	
4	100	DAS2: USER	as in DAS2 traces	6909476	1000
				12274286	
	5000			23077573	69120

Table 5 Average runtime of job instances in the simulations for Scenario 4

Eval.	Res. type	Num. jobs	ID	Cuttime	Delay	RND_RUNTIME_AVG	STAT_RUNTIME_AVG
1	R1	100	11	12274286	1000	302.28	122.04
2	R1	100	11	23077573	69120	19650.95	128.05
3	R1	5000	11	23077573	69120	17548.03	6651.79
4	R1	5000	11	6909476	69120	37764.43	8972.83
5	R1	100	117	12274286	1000	351.39	259.99
6	R1	5000	117	12274286	1000	2820.46	2815.53
7	R1	5000	117	6909476	69120	38614.47	36228.38
8	R2	100	11	23077573	69120	2880.91	107.13
9	R2	100	11	6909476	69120	9694.18	105.12
10	R2	5000	11	23077573	69120	3512.61	2734.65
11	R2	5000	11	6909476	1000	15435.01	13734.96
12	R2	5000	11	6909476	69120	17116.31	10682.18
13	R2	100	117	6909476	69120	22493.29	104.15
14	R2	5000	117	12274286	1000	17959.22	16964.48



(a) Jobs: 10, runtime: 100, wlruntime: 100, Sharcnet

(b) Jobs: 10, runtime: 10000, wlruntime: 10000, Sharcnet

(c) Jobs: 100, runtime: 10000, wlruntime: 10000, Sharcnet

(d) Jobs: 1000, runtime: 1000, wlruntime: 1000, Sharcnet

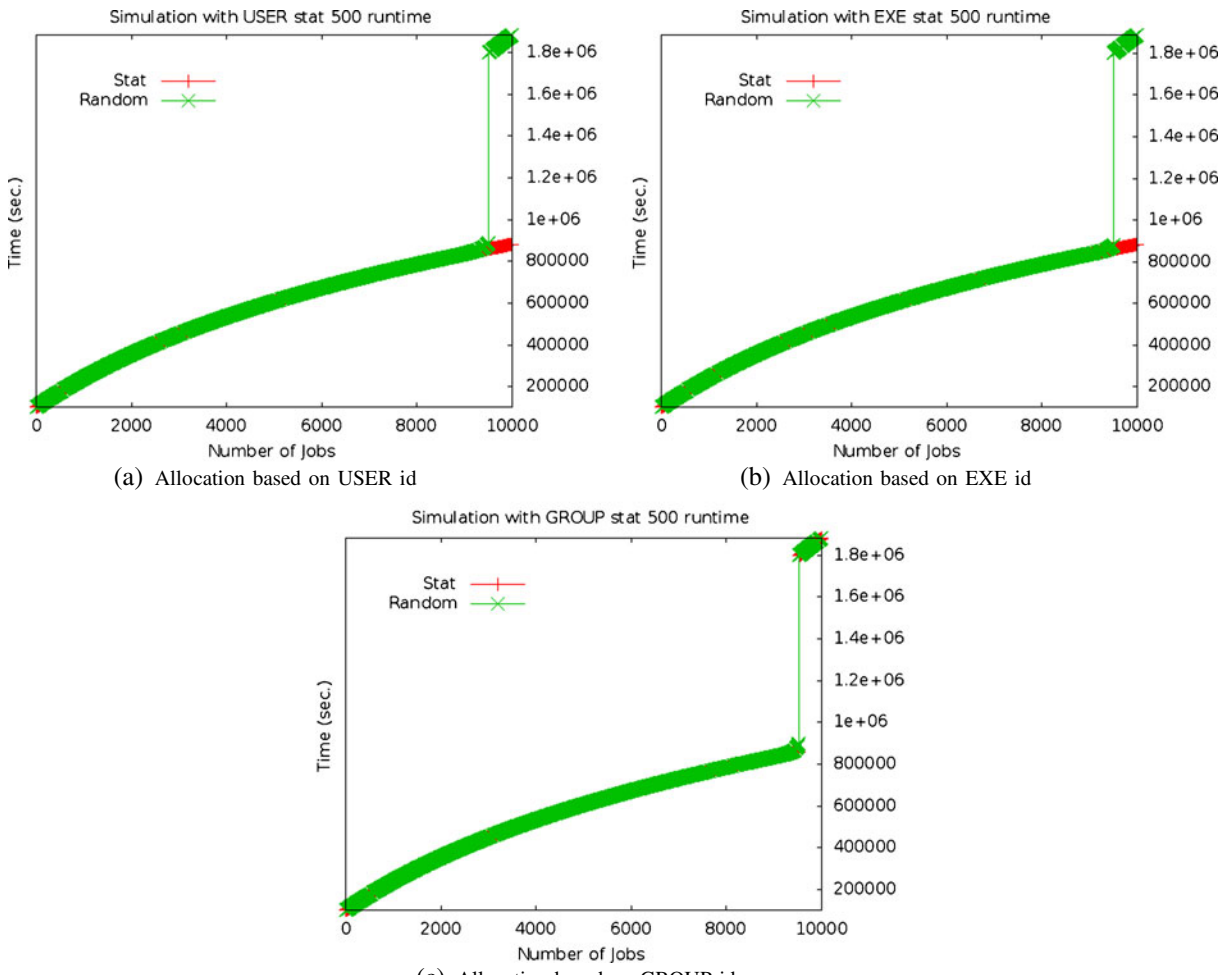
Fig. 6 Plots for Scenario 1

the *PartitionID* fields, which denotes the machine the jobs had been submitted to. We used these partitioned files to feed the simulated resources with background (workload) jobs. In this way we created 10 files from SHARCNET, which we can use to perform simulations on 10 resources. From DAS2, we created 5 files, which we divided into 4 parts based on the logged time equally, to get 20 files to be used for simulating 20 resources.

We have created a statistical job runtime categorization based on the ids and the exact runtimes found in the traces. For each id category (i.e. execution, user and group) we have created a database that contains the calculated mean runtime for all jobs of the considered trace file with the same id. We used these values in the simulations to estimate the runtime of a given job by the broker. In this way the broker (ie. 'GridSimStatQueueBroker' entity in the simulator) knows an estimated waiting time for all available resources in the system based on these statistics at a given time in the simulation, while the resources execute the jobs based on the exact runtimes read from the original traces. During the simulations the runtime of the parameter study jobs are given explicitly, or with an id that refers to the statistically calculated execution time (e.g. see the third columns of Tables 4 and 5).

5.2.2 Simulation results

We have executed numerous simulation runs with all combinations depicted in Table 4. The follow-

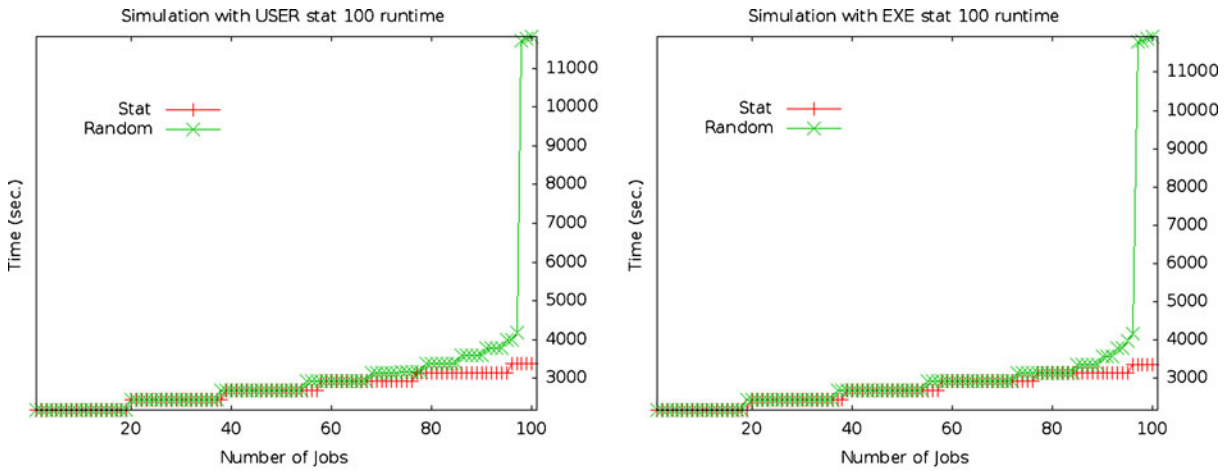


(a) Allocation based on USER id

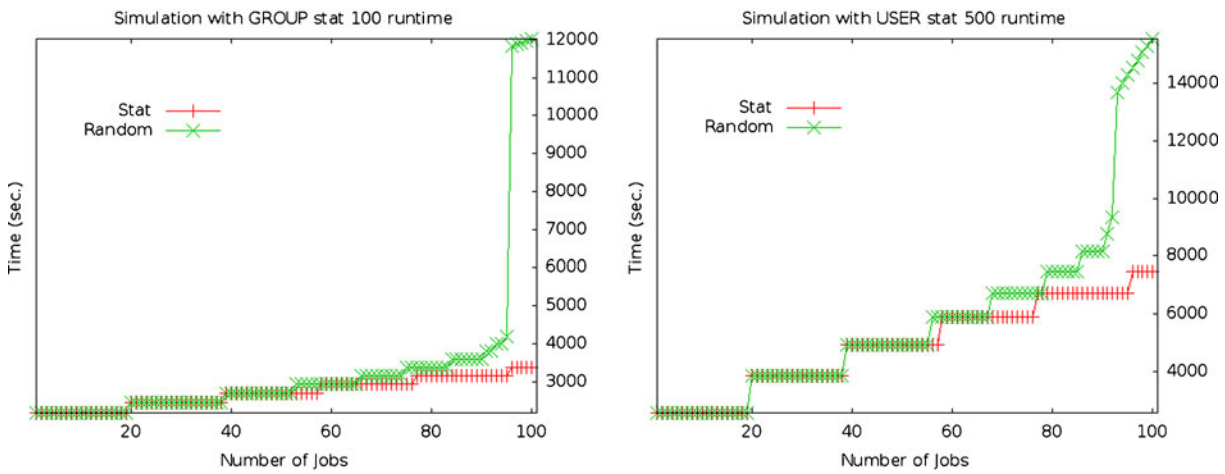
(b) Allocation based on EXE id

(c) Allocation based on GROUP id

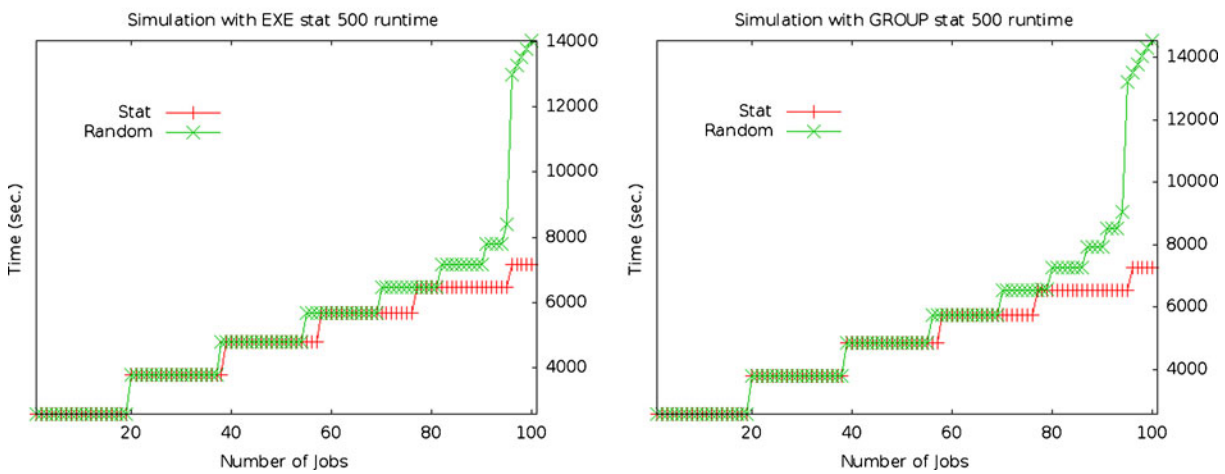
Fig. 7 Plots with detailed job execution times for Scenario 3: 1000 jobs with 500 runtime, DAS2 workloads



(a) Allocation based on USER id (b) Allocation based on EXE id

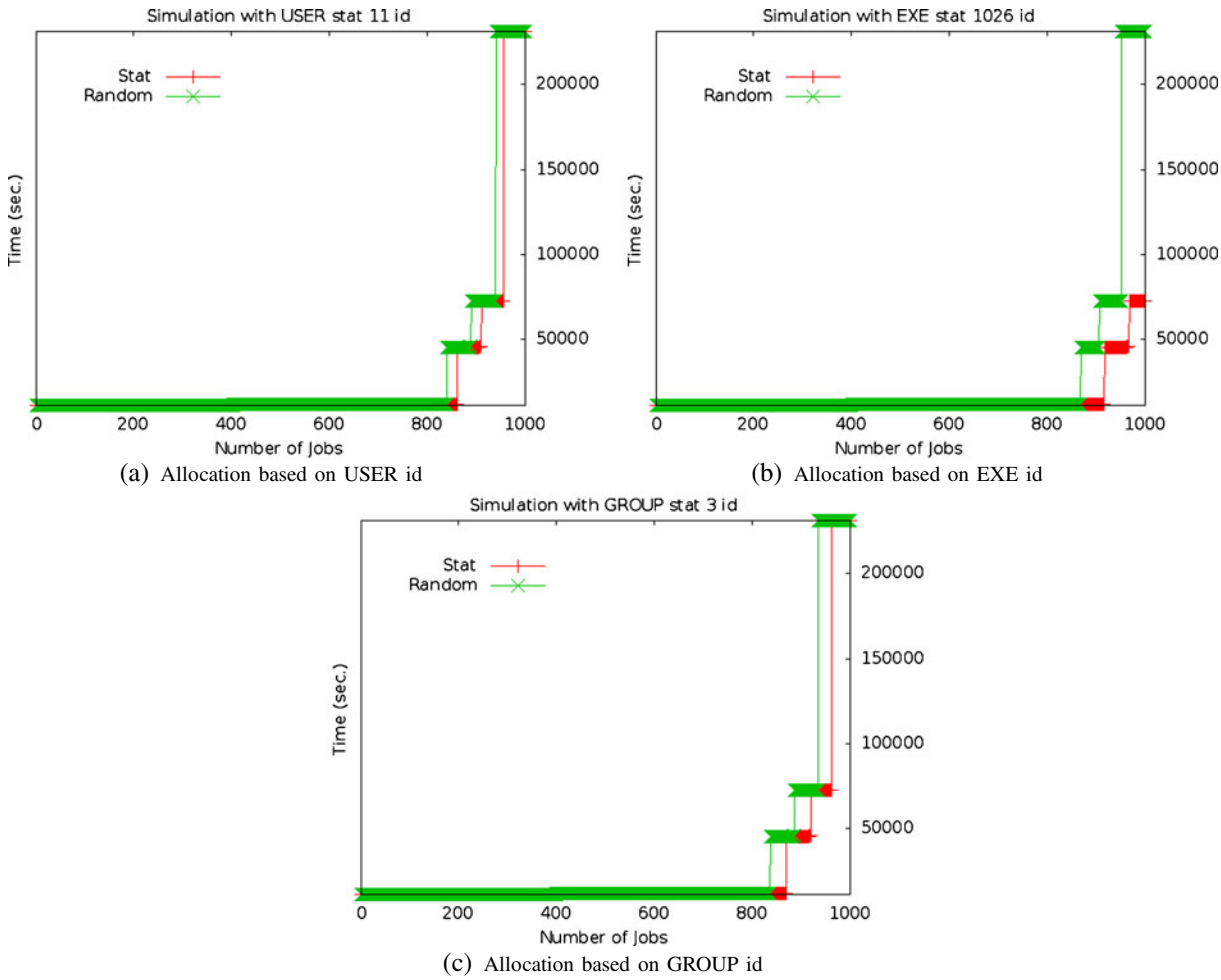


(c) Allocation based on GROUP id (d) Allocation based on USER id



(e) Allocation based on EXE id (f) Allocation based on GROUP id

Fig. 8 Plots with detailed job execution times for Scenario 3: 100 jobs with 100 (a–c) and 500 (d–f) runtime, DAS2 workloads



(a) Allocation based on USER id

(b) Allocation based on EXE id

(c) Allocation based on GROUP id

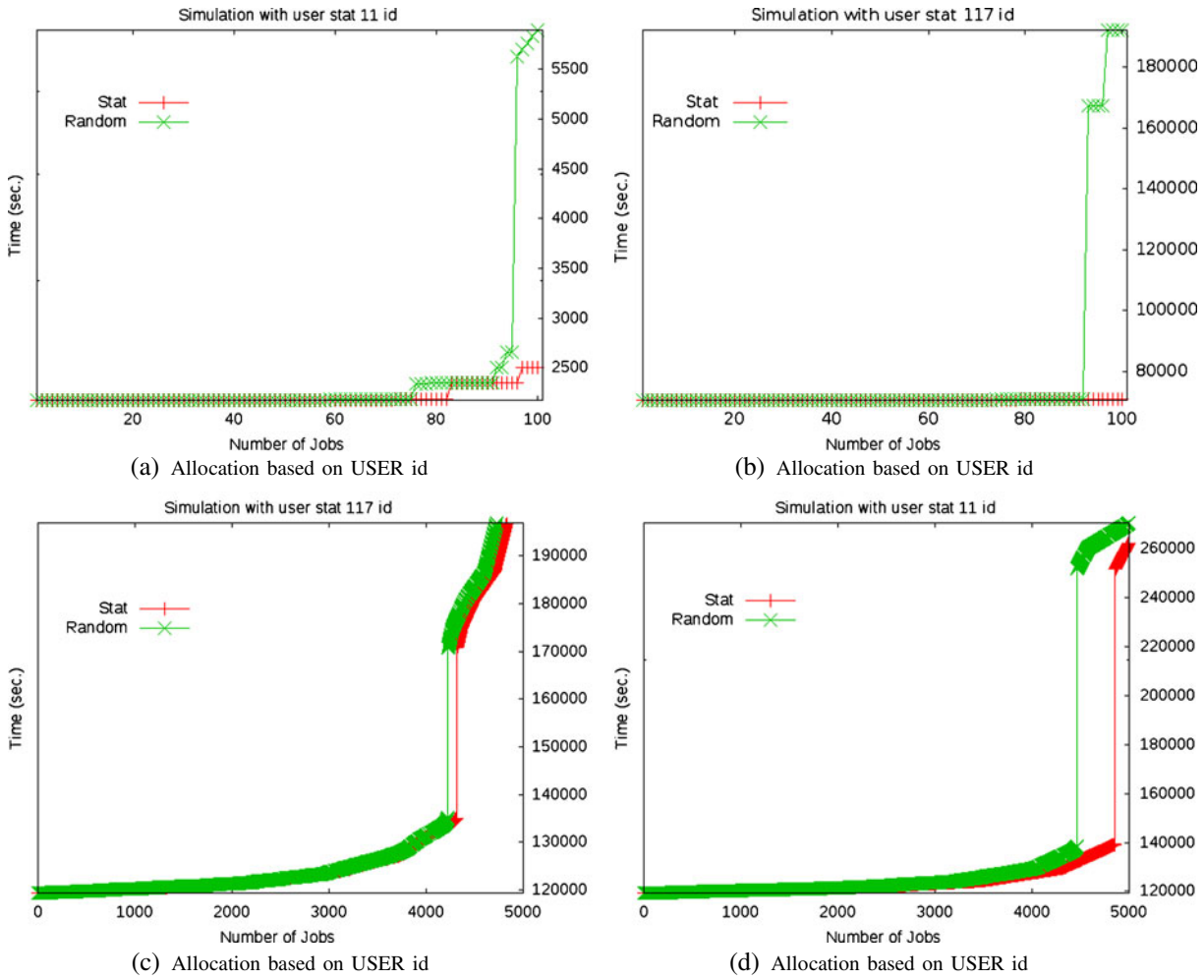
Fig. 9 Plots with detailed job execution times for Scenario 3: 1000 jobs with close runtimes, DAS2 workloads

ing figures show the results of our evaluation. Regarding the first row of the parameter setup table, Fig. 6 shows that as discussed before our algorithm finds the optimal allocation of the jobs, therefore it always performs better than random resource selection.

Figures 7 and 8 show evaluations from the second row of Table 4, while Fig. 9 shows evaluations from the third row of Table 4.

From these plots we can see that our allocation algorithms relying on statistical job runtime categorization read from trace files can perform significantly better than random resource selection. This proves that the categorization is valid in the trace files. On the other hand, there is no significant differences in performance among the groups. In the considered DAS2 trace file the GROUP category has 12, the USER category has 333, and the EXE category has 9070 members. We can see in Figs. 7 and 9 that using the GROUP categorization causes performance loss, but in Fig. 8 we cannot see significant differences.

Figures 10 and 11 show evaluations from the fourth row of Table 4. For these most realistic simulations we used jobs with ids 11 and 117, which have mean execution times 462 and 5801 seconds respectively (computed from the whole trace files). In these cases the simulated Grid environment consisted of resources having various number of machines and processors. For resource setup R1 we used 20 resources, out of which 5–5 resources had 1, 2, 3 and 4 machines respectively,



(a) Allocation based on USER id (b) Allocation based on USER id

(c) Allocation based on USER id (d) Allocation based on USER id

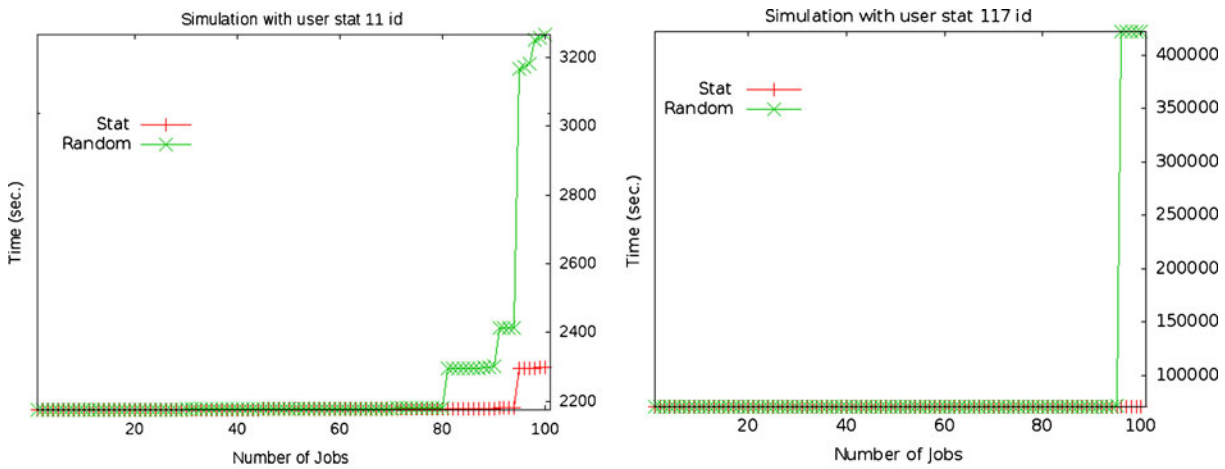
Fig. 10 Plots with detailed job execution times for Scenario 4: 100 and 5000 jobs, R1 resource setup, DAS2 workloads

and each machines had 2 processors. Regarding resource setup R2 we used 20 resources, out of which 5-5 resources had 2, 4, 8 and 10 machines respectively, and each machines had 2 processors is this case, too. Besides the figures, we gathered the average runtime of job instances in some of these measurements in Table 5 to better exemplify the differences of the random and statistics-based allocations.

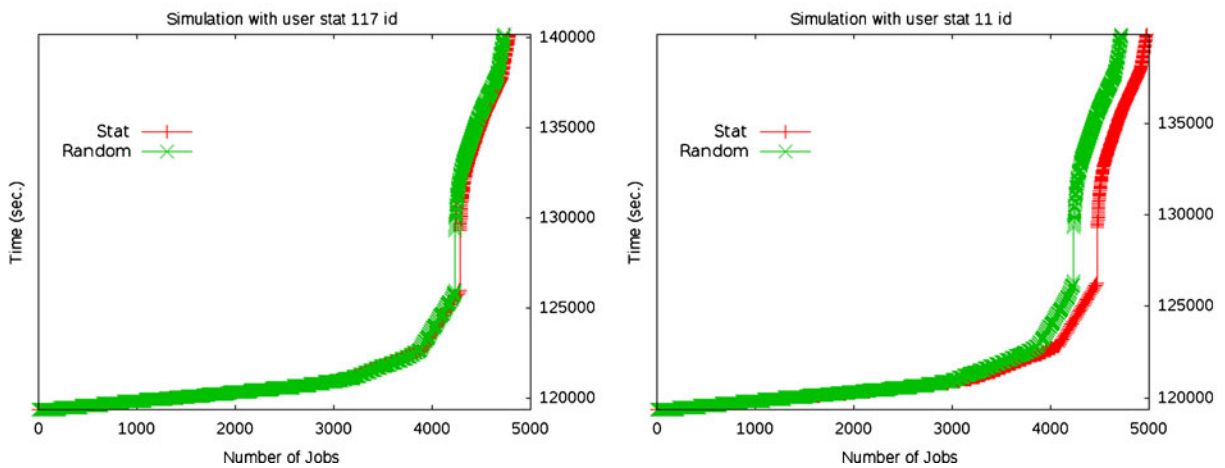
The evaluation of Scenario 4 shows similar results to the simplified Scenario 3. Our proposed statistical brokering approach always performed better than randomized resource selection, and we experienced significant deviances (e.g. see the 2nd, the 4th, the 9th and the 13th rows of Table 5 and Fig. 10d), when we enlarged the resource heterogeneity by varying the number of processors within the machines of a resource (or DCI).

Nevertheless, in some cases we measured very similar makespan (e.g. see the 6th, the 7th and the 14th rows of Table 5 and Fig. 11c), which is due to the inaccuracy of job runtime estimation (based on the mean values of runtimes found in the traces).

Overall, we can state that although the values of the job runtimes sometimes vary highly within the same group categories, estimations by statistically reusing this information from historical trace files can provide us reliable information to perform better allocations then randomized algorithms with no apriori knowledge on job runtimes.



(a) Allocation based on USER id (b) Allocation based on USER id



(c) Allocation based on USER id (d) Allocation based on USER id

Fig. 11 Plots with detailed job execution times for Scenario 4: 100 and 5000 jobs, R2 resource setup, DAS2 workloads

Our future work will address the evaluation of the proposed algorithms in the workflow execution environment [14] of the SHIWA project [23], and the examination of the applicability of this approach to Cloud-based DCIs. To this end we will investigate how virtualization affects the execution time of previously developed workflow applications, and how comparisons can be made among Grid and Cloud resources for multi-job allocations.

6 Conclusion

In this paper we addressed the multi-job allocation problem in distributed systems. In order to cope with the high uncertainty and unpredictable load of these infrastructures, we proposed a statistics-based brokering approach for allocating multi-job instances among resources of multi-Grid systems consisting of several Distributed Computing Infrastructures. Compared to previous studies showing that job runtime predictions of users are inaccurate and unusable for Grid scheduling, our aim was to examine the efficiency of using statistical trace data for a similar purpose. We have evaluated our proposed algorithms in two different simulators and found that a multi-job allocation approach using statistical trace data can perform significantly better than randomized resource selection. Moreover, for small deviation, the algorithm obtains the optimum. Its perfor-

mance depends on the background workload and on the variance of job categorization found in the real-world traces.

Acknowledgements The research leading to these results has received funding from the SCI-BUS FP7 project under grant agreement 283481, and from the ER-Flow FP7 project under grant agreement 312579, and it was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TAMOP 4.2.4. A/2-11-1-2012-0001 'National Excellence Program'.

References

1. Constantini, A.: RWavePR workflow at GASuC. Online: http://www.lpds.sztaki.hu/gasuc/index.php?-m=7&s=12 (2012). Accessed 1 Oct 2012
2. Wiggins, A.: Success-Abandonment-Classification workflow at myExperiment. Online: http://www.myexperiment.org/workflows/140.html (2012). Accessed 1 Oct 2012
3. Buyya, R., Murshed, M., Abramson, D.: Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. In: Journal of Concurrency and Computation: Practice and Experience, pp. 1175–1220 (2002)
4. Casanova, H., et al.: Heuristics for scheduling parameter sweep applications in Grid environments. In: Proceedings 9th Heterogeneous Computing Workshop, (HCW 2000). IEEE, Press, Piscataway (2000)
5. Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauve, J., Silva, F.A.B., Barros, C.O., Silveira, C.: Running bag-of-tasks applications on computational Grids: the mygrid approach. In: International Conference on Parallel Processing, pp. 407–416. IEEE Press, Piscataway (2003)
6. Da Silva, D.P., Cirne, W., Vilar Brasileiro F.: Trading cycles for information: using replication to schedule bag-of-tasks applications on computational Grids. Euro-Par 2003 Parallel Processing, pp. 169–180. Springer Berlin Heidelberg (2003)
7. European Grid Infrastructure. Online: http://www.egi.eu/ (2012). Accessed 1 Oct 2012
8. Garey, M.R., Johnson D.S.: Computers and Intractability; a Guide to the Theory of Np-Completeness. W. H. Freeman & Co., New York (1979)
9. Goble, C.A., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Newman, D., Borkum, M., Bechhofer, S., Roos, M., Li, P., De Roure, D.: myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic. Acids Res. **38**(suppl 2), W677–W682 (2010)
10. The Grid Workloads Archive website. Online: http://gwa.ewi.tudelft.nl (2010). Accessed 1 Oct 2012
11. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., Gonzalez-Garcia, J.L., Roblitz, T., Ramirez-Alcaraz, J.M.: Multiple workflow scheduling strategies with user run time estimates on a Grid. J. Grid Comput. **10**(2), 325–346 (2012)
12. Howell, F., McNab, R.: SimJava: a discrete event simulation library for Java. In: Proc. of the International Conference on Web-Based Modeling and Simulation, San Diego, USA (1998)
13. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.J.: The Grid workloads archive. Futur. Gener. Comput. Syst. **24**(7), 672–686 (2008)
14. Kacsuk, P., Farkas, Z., Kozlovszky, M., Hermann, G., Balasko, A., Karoczkai, K., Marton, I.: WS-PGRADE/gUSE Generic DCI gateway framework for a large variety of user communities. J. Grid Comput. **9**(4), 479–499 (2012)
15. Kwok, Y-K., Ahmad. I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. (CSUR) **31**(4), 406–471 (1999)
16. Lee, C.B., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? Springer LNCS, vol. 3277, pp. 253–263 (2005)
17. Maheswaran, M., Ali, S., Siegal, H.J., Hensgen, D., Freund, RF.: Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In: Proceedings Heterogeneous Computing Workshop, (HCW'99), pp. 30–44. IEEE (1999)
18. Parallel workloads archive website. Online: http://www.cs.huji.ac.il/labs/parallel/workload (2009). Accessed 1 Oct 2012
19. Ramirez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., Gonzalez-Garcia, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run-time estimates for online scheduling in hierarchical Grids. J. Grid Computing **9**(1), 95–116 (2011)
20. Oprescu, A., Kielmann, T.: Bag-of-Tasks Scheduling under Budget Constraints. CloudCom, pp. 351–359 (2010)
21. Saha, D., Menasce, D., Porto, S.: Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. J. Parallel Distrib. Comput. **28.1**, 1–18 (1995)
22. Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in Grids. In: 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), pp. 1–10 (2008)
23. SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs (SHIWA) Eu FP7 project. Online: http://www.shiwa-workflow.eu/ (2012). Accessed 1 Oct 2012
24. Building a European Research Community through Interoperable Workflows and Data (ER-flow) Eu FP7 project. Online: http://www.erflow.eu/ (2013). Accessed 1 Oct 2012
25. Silberstein, M., Sharov, A., Geiger, D., Schuster, A.: GridBot, execution of bags of tasks in multiple Grids. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09) (2009)
26. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)