Procedia Computer Science
Volume 29, 2014, Pages 831–843
ICCS 2014. 14th International Conference on Computational Science

Preference-Based Fair Resource Sharing and Scheduling Optimization in Grid VOs

Victor Toporkov[1], Anna Toporkova[2], Alexey Tselishchev[3], Dmitry Yemelyanov[1], and Petr Potekhin[1]

[1] National Research University "MPEI", ul. Krasnokazarmennaya, 14, Moscow, 111250, Russia
{ToporkovVV, YemelyanovDM, PotekhinPA}@mpei.ru
[2] National Research University Higher School of Economics, Moscow State Institute of Electronics and Mathematics, Bolshoy Trekhsvyatitelsky per., 1-3/12, Moscow, 109028, Russia
atoporkova@hse.ru
[3] European Organization for Nuclear Research (CERN), Geneva, 23, 1211, Switzerland
Alexey.Tselishchev@cern.ch

Abstract
In this paper, we deal with problems of efficient resource management and scheduling in utility Grids. There are global job flows from external users along with resource owners' local tasks upon resource non-dedication condition. Competition for resource reservation between independent users, local and global job flows substantially complicates scheduling and the requirement to provide the necessary quality of service. A meta-scheduling model, justified in this work, assumes a complex combination of job flow dispatching and application-level scheduling methods for jobs, as well as resource sharing and consumption policies established in virtual organizations (VOs) and based on economic principles. A solution to the problem of fair resource sharing among VO stakeholders with simulation studies is proposed.

Keywords: Grid, scheduling, slot, job, backfilling, simulation

1 Introduction

Execution of large parallel jobs in distributed computational environments requires co-allocation of significant resources partially shared with their owners. Resource management and job scheduling economic models proved to be efficient in such conditions [10, 5, 9, 11].

Application-level scheduling, as a rule, does not imply any global resource sharing or allocation policy. Resource brokers are usually considered as mediators between users and resource owners. There are a lot of projects belonging to this trend, namely AppLeS, APST, Legion, DRM, Condor-G, Nimrod/G and others. Scheduling and resource management systems in this approach are well-scalable and application-oriented. However, simultaneous scheduling with

doi:10.1016/j.procs.2014.05.075

diverse optimization criteria set by independent users, especially upon possible competition between applications, may deteriorate such quality of service (QoS) characteristics of a distributed environment as total job batch execution time or overall resource utilization.

The formation of a virtual organization (VO) in Grid usually supposes job-flow scheduling. A meta-scheduler or a meta-broker are considered as intermediate chains between the users and local resource management and job batch processing systems. VOs, from one hand, naturally restrict the scalability of resource management systems. (Though, it is worth remarking here, that there is a good experience of enabling interoperability among meta-schedulers belonging to different VOs [14].) On the other hand, uniform rules of resource sharing and consumption, in particular based on economic models, make it possible to improve the job-flow level scheduling and resource distribution efficiency. The "convergence" idea of application-level and job-flow scheduling approaches was declared in relatively early works [12, 16]. Nevertheless, in some well-known models of distributed computing with non-dedicated resources, only the first fit set of resources is chosen depending on the environment state, while job scheduling optimization mechanisms are usually not supported [6, 2, 8]. The aspects related to the specifics of environments with non-dedicated resources, particularly dynamic resource loading, the competition between independent users, users' global and owners' local job flows, are not presented in other models [5, 9, 12].

A meta-scheduling model in VOs proposed in this work differs from known solutions by combining methods of independent job flow management and application-level scheduling [18]. Our contribution is three-fold. First, we justify a model for matchmaking of VO stakeholder's preferences. In contrast to well-known models, the proposed approach assumes job flows and batches formations according to job requests, users', resource owners' and VO administrators' preferences, and further job batch cyclic scheduling based on dynamically updated VO policies, strategies and restrictions. Job batch schedules are optimized by criteria according to the resource sharing and consumption policy established in the VO. Second, we address a problem of early resources releases and rescheduling "on the fly" combining our original cyclic scheduling scheme (CSS) [18] and backfilling [1]. Third, we analyze consistency of schedules based on user runtime estimates. For the overall job-flow execution optimization and a resource occupation time prediction existing schedulers rely on the time specified in the job request, e.g. using Job Submission Description Language (JSDL). However, the reservation time is usually based on user inaccurate runtime estimates. In case, when the application is completed before the term specified in the job request, the allocated resources remain underutilized.

Thus, we outline two main job-flow optimization directions. In the first of them, the optimal or suboptimal scheduling under a given criterion or criteria specified in VO, is performed on the basis of a priori information about local schedules of computational nodes and the resource reservation time for each job execution. CSS belongs to this type of systems. Another approach represents scheduling "on the fly" depending on a dynamically updated information about resource utilization. In this case, schedulers are focused on overall resources load maximization and job start time minimizing. Backfilling may be related to this type of scheduling.

The rest of the paper is organized as follows. Section 2 is devoted to brief analysis of related works. In Section 3, we discuss restrictions of CSS and introduce main requirements for a model of scheduling and fair resource sharing, representing the CSS generalization. Section 4 contains a simulation framework description, variables and parameters for the model of scheduling and fair resource sharing studies, and simulation results. Finally, section 5 summarizes the paper and describes further research topics.

2 Related Works

The scheduling and resource selection problems in Grid are NP-hard due to their combinatorial nature. Many algorithms, heuristic-based solutions, and their combinations have been proposed for parallel jobs and tasks with dependencies in distributed environments.

First fit resource selection algorithms assign any job to the first set of slots matching the resource request conditions without any optimization (backtrack [2] and NorduGrid [8]). Slots are time spans during which the relate CPU node is idle and ready for execution a part of a parallel job. NWIRE system performs a slot "window" allocation under the maximum total execution cost constraint [9]. However, the optimization occurs only on the stage of the best found offer selection. Preference-based matchmaking is not focused on the scheduling process [6]. The job is scheduled on the first available resource according with user's preferences. An approach to resource matchmaking among VOs combining hierarchical and peer-to-peer models of meta-schedulers is proposed in [14].

The co-allocation algorithms described in [10, 11, 7, 15, 4] suppose an exhaustive search and some of them are based on a linear integer programming (IP) [11, 15] or mixed-integer programming (MIP) model [4].

In [3], architecture and an algorithm are presented for performing Grid resources co-allocation without the need for advance reservations based on synchronous queuing of subtasks. However, advance reservation is effective to improve the co-allocation QoS. An online algorithm for co-allocating resources that provides support for advance reservations is proposed in [7]. The co-allocation algorithm presented in [15] uses the 0-1 IP model with the goal of creating reservation plans satisfying user resource requirements. Linear IP-driven algorithms are proposed in [10, 11]. They combine the capabilities of IP and a genetic algorithm, and allow obtaining the best meta-schedule that minimizes the combined cost of all independent users in a coordinated manner. In [4], the authors propose a MIP model which determines the best scheduling for all the jobs in the queue in environments composed of multiple clusters that act collaboratively. The scheduling techniques proposed in [11, 7, 15, 4, 13] are efficient compared with other scheduling techniques under given criteria: the processing cost, the overall makespan, resources utilization, load balancing for tasks with dependencies, etc. However, complexity of the scheduling process is extremely increased by the resources heterogeneity and the co-allocation process, which distributes the tasks of parallel jobs across resource domain boundaries. The degree of complexity may be an obstacle for on-line use in large-scale distributed environments.

In this work, we use algorithms for efficient slot selection based on users', resource owners' and VO administrators' defined criteria with the linear complexity on the number of all available time-slots during the scheduling interval denoting how far in the future the system may schedule resources [18]. Besides, in our approach the job start, finish and execution time for slot search algorithms may be considered as preferences specified by users in accordance with the job total allocation cost. It makes an opportunity to perform more flexible scheduling solutions. An optimization technique for slot combination selection with dynamic 0-1 MIP methods was proposed in [17].

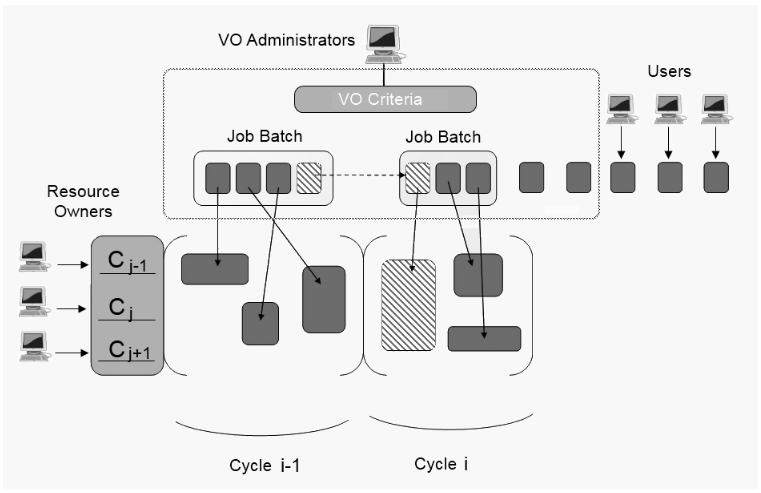

Figure 1: Job flow cyclic scheduling

3 Concept of the scheduling and fair resource sharing model

3.1 Cyclic scheduling scheme

CSS was proposed for a model based on a hierarchical job-flow management system [18]. Job-flow scheduling is performed in cycles by separate job batches on the basis of dynamically updated local schedules of computational nodes (Fig. 1). Sets of available slots and their costs (C_{j-1}, C_j, C_{j+1} in Fig. 1) determined by resource owners are updated based on the information from local resource managers or job batch processing systems. Thus, during every scheduling cycle two problems have to be solved. First of all, the alternative sets of slots that meet the requirements (resource, time, and cost) should be selected. Each alternative is characterized by the total execution cost, runtime, start time, finish time and other parameters (for example power consumption). Second, a combination of alternatives that would be the efficient or optimal in terms of the whole job batch execution in the current scheduling cycle is chosen according to the VO policy.

Let S_i be the family of appropriate sets of slots for executing job i, $i = 1, ..., n$, in the batch, $s_j \in S_i$ be the set of slots that are appropriate by the resource request, the cost $c_i(s_j)$ and the execution time $t_i(s_j), j = 1, ..., N, N = |\bigcup_{i=1}^{n} S_i|$. Denote by S the family of appropriate sets of slots and by $\overline{s} = (s_1, ..., s_n), \overline{s} \in S$, the sequence, which we call the combination of slots, for executing the batch of jobs. Let $f_i(s_j)$ be a function determining the efficiency of executing job i in the batch on the set of slots s_j under the admissible expenses specified by the function $g_i(s_j)$. For example, $f_i(s_j) = c_i(s_j)$ is the price of using the set s_j for the time $g_i(s_j) = t_i(s_j)$. The expenses are admissible if $g_i(s_j) \leq g_i \leq g^*$, where g_i is the level of the total expenses for the execution of a part of jobs from the batch (for example, jobs $i, i+1, ..., n$ or $i, i-1, ..., 1$) and g^* is the restriction for the entire set of jobs (in particular, the restriction on a total time t^* of slot occupation or a limitation on a budget b^* of the virtual organization).

Formally, the statement of the problem of the optimal choice of a slot combination $\overline{s} =$

$(s_1, ..., s_n)$ is as follows:

$$\operatorname*{extr}_{\overline{s} \in S_i} f(\overline{s}) = \operatorname*{extr}_{s_j \in S_i} \sum_{i=1}^{n} f_i(s_j), g_i(s_j) \le g_i \le g^*, g^* = \sum_{i=1}^{n} g_i^0(s_j), \tag{1}$$

where $g_i^0(s_j)$ is the resource expense level function of executing the batch.

The recurrences for finding the extremum of the criterion in (1) for the set of slots $s_j \in S_i, i = \overline{1, n}, j \in \{1, ..., N\}$ based on backward recursion are

$$f_i(g_i) = \operatorname*{extr}_{s_j \in S_i} \{f_i(s_j) + f_{i+1}(g_i - g_i(s_j))\}, g_i(s_j) \le g_i \le g^*, g^* = \sum_{i=1}^{n} g_i^0(s_j), i = \overline{1, n} \tag{2}$$

$$f_{n+1}(g_{n+1}) \equiv 0, g_i = g_{i-1}(s_k), 1 < i \le n, g_1 = g^*, s_k \in S_{i-1},$$

where g_i are the total expenses (utilization time or cost) for using the slots for jobs $i, i+1, ..., n$ of this batch.

The optimal expenses are determined from the equation

$$g_i^*(s_j) = \operatorname*{arg\,extr}_{g_i(s_j) \le g_i} f_i(g_i), i = \overline{1, n}. \tag{3}$$

The optimal set of slots $s_i^* \in \{1, ..., N\}$ in the scheme (2), (3) is given by the relation

$$s_i^* = \operatorname*{arg\,extr}_{s_j \in S_i} f_i(g_i^*(s_j)), i = \overline{1, n}. \tag{4}$$

Here (4) represents the solution of the problem (1). An example of a resource expense level function in (1) is $t_i^0(s_j) = \left\lceil \sum_{s_j} t_i(s_j)/l_i \right\rceil$, where l_i is the number of admissible (alternative) sets of slots for the execution of job i, $\lceil . \rceil$ is the ceiling of $t_i^0(s_j)$. Then the constraint on the total time of slot occupation in the current scheduling cycle can have the form

$$t^* = \sum_{i=1}^{n} t_i^0(s_j). \tag{5}$$

Let us consider several problems of practical importance.

1. Maximization of profit of resource owners under restrictions on the total time of slot utilization. Suppose it is required to select a set of slots for executing a batch of n jobs so as to maximize the total cost of resource utilization

$$f_i(t_i) = \max_{s_j \in S_i} \left\{c_i(s_j) + f_{i+1}(t_i - t_i(s_j))\right\}, i = 1, ..., n, f_{n+1}(t_{n+1}) \equiv 0. \tag{6}$$

   The restriction on the total time of using slots by all the jobs is given by (5).

2. Minimization of the total completion time of a batch of jobs under a restriction on the budget b^* of the virtual organization:

$$f_i(c_i) = \min_{s_j \in S_i} \left\{t_i(s_j) + f_{i+1}(c_i - c_i(s_j))\right\}, i = 1, ..., n, f_{n+1}(c_{n+1}) \equiv 0. \tag{7}$$

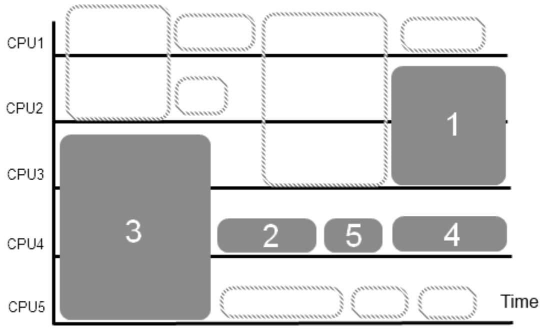

Figure 2: An example of alternatives allocation for a batch of five jobs

3. Minimization of the total cost of executing a batch of n jobs under a restriction on the total time(5) of slot utilization:

$$f_i(t_i) = \min_{s_j \in S_i} \{c_i(s_j) + f_{i+1}(t_i - t_i(s_j))\}, i = 1, ..., n, f_{n+1}(t_{n+1}) \equiv 0. \tag{8}$$

4. Minimization of the idleness of resources under the restriction on the total time of their utilization. On the one hand, the resource owners restrict the time of slot utilization to balance their own (local) and users' job flows. On the other hand, the owners naturally strive to minimize the idleness of resources. Assume that the slot utilization time is bounded by (5). The problem is reduced to finding a set of slots that satisfy this restriction:

$$f_i(t_i) = \max_{s_j \in S_i} \{t_i(s_j) + f_{i+1}(t_i - t_i(s_j))\}, i = 1, ..., n, f_{n+1}(t_{n+1}) \equiv 0. \tag{9}$$

The above functional equations (6-9) are concretizations of (2) and are implemented as simulation environment components [18].

Among the major CSS restrictions in terms of an efficient scheduling and resource allocation one may outline the following. First of all, it is not possible to affect execution parameters of an individual job: the search for particular alternatives is performed on the First Fit principle, while choice the optimal combination (4) represents only the interests of VO upon the whole. Thus, CSS-approach does not take into account user interests and preferences, and therefore obstructs fair resource sharing. Second, the job batch scheduling is based on a user estimation of the particular job runtime $t_i(s_j)$ (often inaccurate). Thus, in case of estimation incorrectness, the early released resources may become idle reducing the distributed environment utilization level. Third, the job batch scheduling requires allocation of a multiple "nonintersecting" in terms of slots alternatives, and at the same time only one alternative is chosen for each job execution.

Fig. 2 shows a job batch scheduling example consisting of five independent jobs. Highlighted rectangles schematically represent all "nonintersecting" in terms of slots alternatives found for the batch on the scheduling cycle in "CPU - Time" space. Filled rectangles represent a combination of the alternatives selected by the metascheduler. Thus, available resources are fragmented, and their utilization level, especially at the beginning of the considered scheduling interval, is relatively low.

The following subsection is dedicated to the CSS generalization and further development.

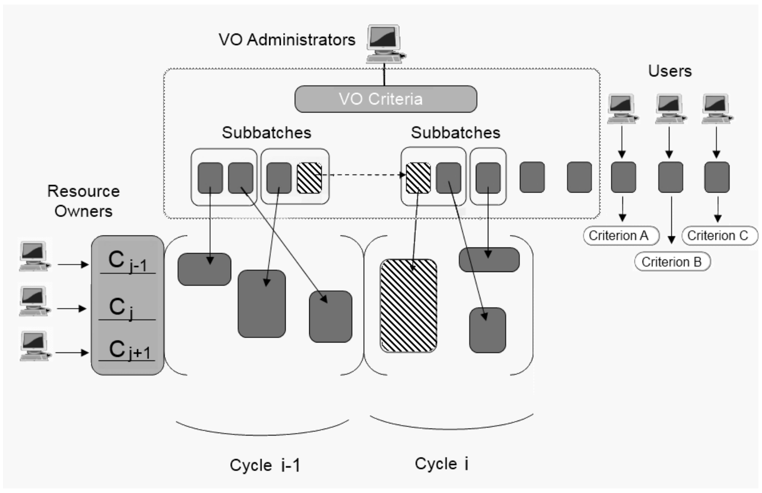

Figure 3: Job flow cyclic scheduling with batch-slicing

3.2 CSS generalization

For the metascheduling concept implementation we put the following requirements for the model of scheduling and fair resource sharing among the VO stakeholders (we name this model as Batch-slicer). First, VO administrators should be able to manage the scheduling process by establishing a job-flow execution policy. Second, VO users should have an opportunity to affect their jobs execution schedule by setting an optimization criterion. Third, resource owners should be able to control utilization level of their computational nodes by specifying their pricing model during the scheduling interval. In order to satisfy the user preferences a desirable optimization criterion is introduced into the resource request format, e.g. JSDL. Unlike the so-called soft constraints [12] representing the user preferences, the optimization criterion defined in the resource request is considered during the stage of alternatives (slot sets) search. An **A**lgorithm searching for **E**xtreme **P**erformance (AEP) described in details in [17] is used to select optimal alternatives under a given criterion: the earliest start/finish time; the minimum total allocation cost/execution runtime.

The next step of the CSS generalization includes the job system formation approach. We propose a separation of the initial job batch into a set of sub-batches and each sub-batch scheduling at the same given scheduling interval. The idea of "slicing" can be particularly noticeable at a relatively high distributed environment resources utilization level. According to the alternatives search algorithm adopted in CSS, the number of execution alternatives for a job batch may be relatively small (up to just a single alternative for every job at a high resource utilization level). Such a small number of alternatives found may affect the optimal slot combination selection, and therefore, may reduce overall scheduling efficiency. The job batch "slicing" increases the number of alternatives found for high-priority jobs and diversifies the choice on the slots combination selection stage, and thereby increases the resource sharing efficiency according to VO policy.

In view of described modifications, the proposed model, named as Batch-slicer, is schematically shown in Figure 3: an optimization criterion is specified for each job, and the job batch is separated to the sub-batches during the scheduling cycle.

3.3 Cyclic scheduling method combined with backfilling

Batch-slicer makes it possible to optimize the job-flow execution according to the VO stakeholders' preferences on condition that a sufficient number of alternatives were found for the batch jobs during the scheduling cycle. Backfilling [1] responds to early resources releases and performs "on the fly" rescheduling which is very important when a user job runtime estimation is significantly different from the actual job execution time. There are some limitations of backfilling for distributed computing. The first one is inefficient resource usage by criteria differed from an average job start time (especially at a relatively low level resources load). The second one is a principal inability to affect the resource sharing quality by defining policies and criteria in VO. Nevertheless it is appropriate to consider the use of backfilling to reschedule tasks [13] and avoid resources fragmentation.

We propose a combined approach. During every scheduling cycle a set of high priority jobs, for example the most "expensive" (by total execution cost) or the most critical in terms of required resource (by performance), is allocated from the initial job batch. These jobs should be scheduled before other jobs, probably, without compliance with the queue discipline. High priority jobs are grouped into a separate sub-batch. The scheduling of this sub-batch is further performed by Batch-slicer based on the preliminary known resources utilization schedule. The scheduling of the rest batch jobs is performed by backfilling with the dynamically updated information about the actual computational nodes utilization. Thus, the cyclic scheduling method combined with backfilling (Batch-slice-Filling - BSF) unites the main advantages of both Batch-slicer and backfilling, namely the optimization of the most time-consuming jobs execution as well as the efficient resource usage, preferential job execution queue order compliance and relatively low response time.

4 Simulation studies

4.1 Simulation environment setup

A series of studies were carried out with the simulation environment [18] in order to investigate the proposed job batch scheduling approaches and to compare them with known scheduling algorithms. The scheduling environment core consists of the following major components: computational procedures and random variable functions implementation for the environment parameters generation; job requests and distributed computing environment generation; AEP slot processing; an algorithm for optimal alternatives combination selection; Batch-slicer, backfilling, and BSF modules.

We introduce some realistic features into our simulation approach. The model components general settings are used for the experiments as follows. A typical scheduling interval length is assumed to be 600 units of time in simulation steps. The number of nodes in the resource domain is equal to 24. The nodes performance level is given as a uniformly distributed random value in the interval [2, 10]. This configuration provides a sufficient resources diversity level while the difference between the highest and the lowest resource performance levels will not exceed one order within a particular resource domain. Uniform distribution was chosen in the assumption that the CPU node composition is formed by resource selection based on such characteristics as a CPU node type, performance, locations, etc. (hard constraints according to [12]).

The node prices are assigned during the pricing stage depending on the node performance level and a random "discount /extra charge" value which is normally distributed. The number of user jobs in each scheduling cycle is assumed to be 20.

Criterion	N_A	Start time	Execution time	Finish Time	Cost
Start time	12.8	171.7	56.1	227.8	1281.1
Execution time	10.6	214.5	**39.3**	253.9	1278.5
Finish time	12.2	**169.6**	45	**205.5**	1283.2
Cost	12.9	262.6	55.5	318	**1098.3**
CSS	12.1	222	50.3	272.3	1248.4

Table 1: Scheduling results with VO users' preferences

The jobs budget limit is generated in such a way that the "richest" users can afford to use "expensive" resources with the price formed as a "market value + 60% extra charge", and the "poorest" users have been forced to rely on 60% discounts. These factors prevent the monopoly for the most expensive and, therefore, the high-performance resources.

4.2 Results

Table 1 shows the results of individual jobs scheduling depending on the optimization criterion specified by the user: job start and finish time, execution time in steps of the simulator [18], and cost (AEP minimizes the value of the specified criterion). The choice of one of the four optimization criteria is made randomly with uniform distribution at the stage of job batch generation. Uniform distribution is used because no prevalent optimization criterion can be chosen. The last row of Table 1 shows the results of scheduling of the same batch job with the initial CSS without optimization at the stage of alternatives search. Simulation of 5000 individual scheduling cycles was conducted. Average number of execution alternatives (N_A in Table 1) found for the jobs during one scheduling cycle almost does not depend on the chosen optimization criterion.

The best values against start and finish time criteria as well as by execution time and cost (the minimal values are marked in bold) are achieved by the jobs for which the corresponding optimization criterion is specified ("Criterion" column). The only exception is the minimum finish time strategy: the jobs for which this optimization criterion was specified show on average not only the minimal finish time, but also the minimal start time. On average the use of an optimization criterion in Batch-slicer, in comparison with CSS, when executing individual jobs, allows reducing job start and finish time by more than 23%, reducing execution time by 21% and reducing execution cost by 12%.

The individual jobs scheduling results show that users can affect the execution of their own jobs by specifying an optimization criterion. This is achieved due to the fact that in the presence of different requirements to efficiency of job execution resources are allocated among the jobs unevenly, depending on the criterion used in selection. Note that in the initial CSS at the stage of alternatives search all resources are allocated among the jobs uniformly.

The next experiment is dedicated to comparing the scheduling results when slicing the initial job batch in Batch-slicer into sub-batches at different levels of environment utilization. When choosing the optimal execution alternatives combination the average job execution time T_{CPU} minimization problem was being solved. Total slot utilization time for an alternative is determined as the sum of slot lengths being part of the composed "window".

Figure 4 shows the value of T_{CPU} depending on the number of sub-batches $k \in \{1, 2, 3, 5, 6, 10, 20\}$. When performing the series of experiments the environment utilization level is determined by the relative average number of failures Y - scheduling cycles in the course of which the execution schedule for all the batch jobs was not found.

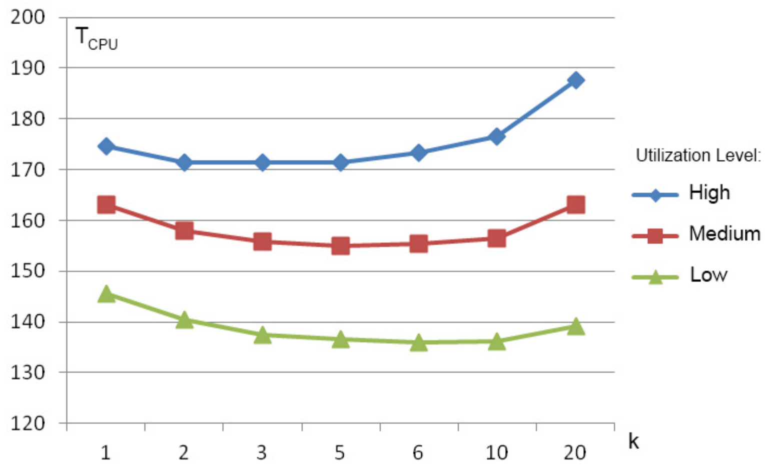

Figure 4: Batch jobs execution time T_{CPU} depending on the number of sub-bathes k

c	L_c	U	S	Y
2	256.6	0.44	527.1	0
4	234.9	0.39	939.6	0.001
6	185.4	0.31	1112.3	0.013
8	109.8	0.18	878.7	0.024
10	71	0.12	710.3	0.025

Table 2: Scheduling results with VO resource owners' preferences

The experiments were conducted under high ($Y = 0.3$), medium ($Y = 0.03$), and low utilization levels ($Y < 0.0002$). An increase of composed sub-batches number causes an increase of alternatives number for execution an individual job, a decrease of total job execution cost, and an increase of failures Y. When increasing the level of available resources the number of alternatives for an individual job execution increases, the relative number of failures Y decreases, and the total cost of job batch execution decreases.

Thus, Batch-slicer allows not only taking into account VO administrators' preferences (by optimizing at the alternatives set selection stage, like in the initial CSS), but can also provide a better value of the target criterion in comparison with CSS by slicing into sub-batches.

Table 2 shows the scheduling results with Batch-slicer from resource owners' point of view by the example of a single CPU node characteristics depending on the unit cost c, specified for the use of scheduling interval $T = 600$: L_c - total slot utilization time in the scheduling interval, U - average value of relative resource utilization in the scheduling interval, P - average profit made by the resource owner, and Y - relative number of scheduling failures.

As can be seen from Table 2, resource owners are able to control their profit P and the computational node utilization level U in the scheduling interval T by proposing the unit cost c of using their node. Profit extremum is achieved when proposing the cost close to the "average market cost", i.e. the average cost for a resource with similar performance, proposed by other resource owners.

Figure 5 shows batch job average execution time T_{CPU} and average start time T_{start} depending on the ratio according to which slicing into sub-batches was made in BSF: n_{CSS} - the

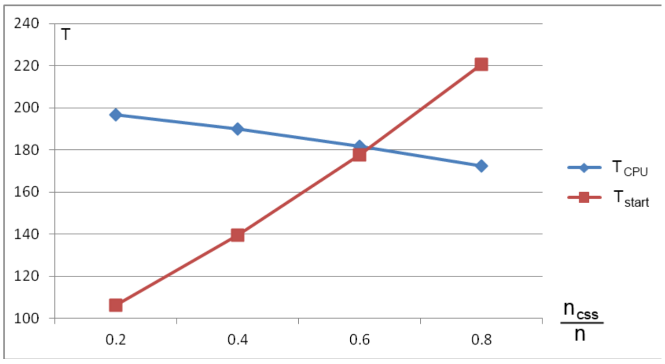

Figure 5: Average job execution T_{CPU} and start T_{start} time in BSF

number of jobs in the first sub-batch, scheduled with CSS, n - the total number of jobs in the batch. Slicing into sub-batches was made on basis of a priority - the order of jobs in the batch, without taking into account the characteristics of the jobs themselves.

As seen from Figure 5, if a major part of the job is scheduled with Batch-slicer then a better value of the target VO scheduling criterion - execution time T_{CPU} - is achieved, but average job start time T_{start} is delayed. And on the contrary, if a major part of the job is scheduled with backfilling, then average start time approaches the beginning of the scheduling interval but the value of the target optimization criterion deteriorates. Particular emphasis should be placed on the cross point of graphs in Figure 5. Its presence given that the graphs are monotone suggests the possibility of reaching a compromise between average start time and the value of the VO target optimization criterion.

Batch-slicer and CSS form preliminary job batch execution schedules in the scheduling interval without taking into account the situations in which real job execution time is less than the time specified by the user. On the other hand, backfilling conducts scheduling on basis of dynamically updated information on job execution status and computational node utilization. Thanks to this it can provide the efficient job flow execution. A simulation was conducted to study and to compare the efficiency of schedules performed with CSS, Batch-slicer and backfilling.

In the simulation real job execution time differed considerably from resource advanced reservation time. Real job execution time was specified as a random variable uniformly distributed in the interval $[0.2 * T_{res}, T_{res}]$, where T_{res} - time reserved for job execution. Uniform distribution is chosen as it is almost impossible to predict real job execution time on the specified resources.

Table 3 contains the average job execution time values (the target optimization criterion) and the average job start time obtained: 1) at the stage of preliminary scheduling based on job execution time estimate T_{res} (”Scheduled” column); 2) as the result of execution simulation of the composed schedule taking into account real job execution time on the chosen resources (”Real” column).

It can be seen from Table 3, that even if the difference between resource reservation time and real job execution time is significant the advantage of Batch-slicer over backfilling against the VO target optimization criterion not only remains but increases. That is because backfilling does not optimize against criteria different from start time and a more compact job location uses almost all available resources including those less advantageous against the target criterion.

Thus, results of this stress test show that preliminary schedules formed in the beginning

Algorithm	Execution time		Start time	
	Scheduled	Real	Scheduled	Real
Backfilling	187.7	115.1	69	37.3
CSS	150.1	90.4	281.2	281.2
Batch-slicer	138.6	83.5	223.8	223.8
Batch-slicer advantage over backfilling	26.2%	27.5%	-69%	-83%

Table 3: Real and scheduled job execution time

of the scheduling cycle are consistent against the criteria determined in VO in the case when real execution time differs significantly from resource reservation time. Note that additional advantage can be achieved by rescheduling with backfilling taking into account the information about computational nodes' current utilization.

5 Conclusions and future work

5.1 Main results

In this work, we address metascheduling and co-allocation strategies with different target criteria and based on scheduling and fair resource sharing model taking into account all VO stakeholders' preferences on the basis of economic principles.

A solution to the problem of fair resource sharing among VO stakeholders is proposed. Based on CSS and backfilling union a combined approach BSF is proposed. The approach shows compromise results compared to Batch-slicer and backfilling. The consistency of scheduling made with Batch-slicer when real job execution time is significantly different from user's estimate is shown.

5.2 Future work

Further research will be related to a more precise investigation of dividing the job flow into sub-batches depending on the jobs characteristics and computing environment parameters as well as to studying the mechanism of rescheduling based on the information about computational nodes current utilization.

5.3 Acknowledgments

This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Leading Scientific Schools (SS-362.2014.9), the Russian Foundation for Basic Research (grant no. 12-07-00042), and by the State Task of the Ministry of Education and Science of the Russian Federation (the basic part, task no. 2014/123, project no. 2268).

References

[1] The Moab adaptive computing suite, last viewed march 2012. `http://www.adaptivecomputing.com/products/moab-adaptive-computing-suite.php`.

[2] Kento Aida and Henri Casanova. Scheduling mixed-parallel applications with advance reservations. In *17th IEEE Int. Symposium on HPDC*, pages 65–74, New York, USA, 2008. IEEE CS Press.

[3] Farag Azzedin, Muthucumaru Maheswaran, and Neil Arnason. A synchronous co-allocation mechanism for Grid computing systems. *Cluster Computing*, 7:39–49, 2004.

[4] Hector Blanco, Fernando Guirado, Josep Lluis Lerida, and V. Albornoz. MIP model scheduling for multi-clusters. *Euro-Par 2012*, 7640:196–206, 2012.

[5] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in Grid computing. *Grid Computing. Concurrency and Computation*, 5(14):1507–1542, 2002.

[6] Massimo Cafaro, Maria Mirto, and Giovanni Aloisio. Preference-based matchmaking of grid resources with CP-nets. *Grid Computing*, 2(11):211–237, 2013.

[7] Claris Castillo, George Rouskas, and Khaled Harfoush. Resource co-allocation for large-scale distributed environments. In *18th ACM International Symposium on High Performance Distributed Computing*, pages 137–150, New York, USA, 2009. ACM.

[8] Erik Elmroth and Johan Tordsson. A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability. *J. of Concurrency and Computation*, 18(25):2298–2335, 2009.

[9] Carsten Ernemann, Volker Hamscher, and Ramin Yahyapour. Economic scheduling in Grid computing. In D. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *JSSPP*, volume 18, pages 128–152. Springer, Heidelberg, 2002.

[10] Saurabh Kumaer Garg, Pramod Konugurthi, and Rajkumar Buyya. A linear programming-driven genetic algorithm for meta-scheduling on utility Grids. *Emergent and Distr. Systems*, (26):493–517, 2011.

[11] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers. *Parallel and Distributed Computing*, 6(71):732–749, 2011.

[12] Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Weglarz. Multicriteria aspects of Grid resource management. In J. Nabrzyski and J. Schopf, editors, *Grid resource management. State of the art and future trends*, pages 271–293. Boston, 2003.

[13] Alexandra Olteanu, Florin Pop, Ciprian Dobre, and Valentin Cristea. A dynamic rescheduling algorithm for resource management in large scale dependable distributed systems. *Computers and Mathematics with Applications*, 9(63):1409–1423, 2012.

[14] Ivan Rodero, David Villegas, Norman Bobroff, Yanbin Liu, Liana Fong, and S. Masoud Sadjadi. Enabling interoperability among Grid meta-schedulers. *Grid Computing*, 2(11):311–336, 2013.

[15] Atsuko Takefusa, Hidemoto Nakada, Tomohiro Kudoh, and Yoshio Tanaka. An advance reservation-based co-allocation algorithm for distributed computers and network bandwidth on QoS-guaranteed Grids. In D. Feitelson and U. Schwiegelshohn, editors, *JSSPP*, volume 6253, pages 16–34. Springer, Heidelberg, 2010.

[16] Victor Toporkov. Application-level and job-flow scheduling: an approach for achieving quality of service in distributed computing. In V. Malyshkin, editor, *PaCT 2009*, volume 5968, pages 350–359. Springer, Heidelberg, 2009.

[17] Victor Toporkov, Anna Toporkova, Alexey Tselishchev, and Dmitry Yemelyanov. Slot selection algorithms in distributed computing with non-dedicated and heterogeneous resources. In V. Malyshkin, editor, *PaCT 2013*, volume 7979, pages 120–134. Springer, Heidelberg, 2013.

[18] Victor Toporkov, Alexey Tselishchev, Dmitry Yemelyanov, and Alexander Bobchenkov. Composite scheduling strategies in distributed computing with non-dedicated resources. *Procedia Computer Science*, (9):176–185, 2012.