Computing and Informatics, Vol. 34, 2015, 45–76

METASCHEDULING AND HEURISTIC CO-ALLOCATION STRATEGIES IN DISTRIBUTED COMPUTING

Victor Toporkov, Dmitry Yemelyanov, Petr Potekhin

National Research University "MPEI"
ul. Krasnokazarmennaya, 14, Moscow, 111250, Russia
e-mail: {ToporkovVV, YemelyanovDM, PotekhinPA}@mpei.ru

Anna Toporkova

National Research University Higher School of Economics
Moscow State Institute of Electronics and Mathematics
Bolshoy Trekhsvyatitelsky per., 1-3/12, Moscow, 109028, Russia
e-mail: AToporkova@hse.ru

Alexey Tselishchev

European Organization for Nuclear Research (CERN)
Geneva, 23, 1211, Switzerland
e-mail: Alexey.Tselishchev@cern.ch

Abstract. In this paper, we address problems of efficient computing in distributed systems with non-dedicated resources including utility grid. There are global job flows from external users along with resource owner's local tasks upon the 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 metascheduling concept, justified in this work, assumes a complex combination of job flow dispatching and application-level scheduling methods for parallel jobs, as well as resource sharing and consumption policies established in virtual organizations and based on economic principles. We introduce heuristic slot selection and co-allocation strategies for par-

allel jobs. They are formalized by given criteria and implemented by algorithms of linear complexity on an available slots number.

Keywords: Distributed computing, economic scheduling, resource management, co-allocation, slot, job, task, batch

Mathematics Subject Classification 2010: AB-XYZ TBA

1 INTRODUCTION

Execution of large parallel jobs in distributed computational environments requires allocation of a significant amount of resources partially shared with their owners [1, 2, 3, 4]. Today well-known algorithms, their combinations and heuristics used by schedulers are usually unable to provide optimal or suboptimal solutions in terms of heterogeneous distributed environments and dynamically changing sets of available computational nodes and their utilization. Resource management and job scheduling economic models proved to be efficient in such conditions [1, 2, 3].

Two established trends may be outlined among diverse approaches to distributed computing. The first one is based on the available resources utilization and application level scheduling. As a rule, this approach does not imply any global resource sharing or allocation policy. Application agents, i.e. resource brokers [5, 6, 7, 8, 9, 10, 11], are usually considered as mediators between the users and the resource owners. There are many projects belonging to this trend, namely AppLeS [6], APST [7], Legion [8], DRM [9], Condor-G [10], Nimrod/G [11] and others.

Another trend is related to the formation of user's virtual organizations (VO) and job flow scheduling [12, 13, 14]. In this case, an external scheduler, e.g. a grid dispatcher, a metascheduler or a Meta-Broker [15], is an intermediate chain between the users and local resource management and job batch processing systems.

Scheduling and resource management systems belonging to the first approach are well-scalable and application-oriented. However, simultaneous application-level scheduling with diverse optimization criteria set by independent users, especially upon possible competition between applications, may deteriorate such integral QoS characteristics of a distributed environment as total job batch execution time or overall resource utilization. VOs, on one hand, naturally restrict the scalability of resource management systems. On the other hand, uniform rules of resource sharing and consumption, in particular based on economic models [1, 2, 3, 4, 16, 17, 18], make it possible to improve the job-flow level scheduling and resource distribution efficiency.

The "convergence" idea of application-level and the job-flow scheduling approach was declared in relatively early works [14, 19, 20, 21]. 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 [22, 23, 24], while

job scheduling optimization mechanisms are usually not supported. The aspects related to the specifics of environments with non-dedicated resources – particularly the dynamic resource loading, the competition between independent users, users' global and owners' local job flows – are not presented in other models [14, 16, 17].

A metascheduling concept in VOs proposed in this work fundamentally differs from known solutions by combining methods of independent job flow management and application-level scheduling [19, 20, 21]. We propose a model of independent job flows management based on economic principles. The job scheduling is performed cyclically for alternative sets of preliminary selected resources (alternatives) [25]. In contrast to well-known models, the proposed approach assumes job flows and batches formations according to job features, characteristics, resource requirements, users' preferences, and further job batch cyclic scheduling based on dynamically updated VO policies, strategies and restrictions. Job batch schedules are optimized by a criteria vector according to the resource sharing and consumption policy established in the VO.

The rest of this paper is organized as follows. Section 2 is devoted to analysis of various VO stakeholders preferences and related works in distributed computing. There is a formal problem statement for a cyclic scheduling scheme. Then we discuss restrictions of this scheme. In Section 3, we introduce main requirements for a model of scheduling and fair resource sharing, representing the cyclic scheduling scheme generalization. A combined scheduling approach based on generalized cyclic scheduling scheme and backfilling is proposed in Section 4. Section 5 contains a simulation framework description, variables and parameters for the model of scheduling and fair resource sharing studies. The simulation results are presented in Section 6. Section 7 focuses on the research of the scheduling method combined with backfilling. Finally, Section 8 summarizes the paper and describes further research topics.

2 SCHEDULING PROBLEMS IN VO

2.1 VO Stakeholders and Their Preferences

The scheduling efficiency in VO may be considered from different points of view. On the one hand, one of the most important indicators is the available resources utilization level and an average job starting time ("response" time). Computational nodes of distributed environments with non-dedicated resources are generally partially utilized by local high priority tasks. Thus, the available resources of VO are represented as a set of slots, i.e. time spans during which the related node is idle and ready for executing a part of a parallel job. The existence of an available slots set with different start and finish times as well as a different performance rate (depending on the CPU node characteristics), complicates the problem of efficient resource co-allocation and job-flow execution in the distributed environment. The resource fragmentation also reduces the overall distributed environment utilization level. On the other hand, the VO scheduling efficiency may be considered in terms of com-

pliance with certain scheduling policies and VO shareholders preferences. Besides, there are contradictory interests of VO users, resources owners and administrators. The users are usually interested in the earliest start time for their applications with the lowest cost, while resource owners intend to obtain the maximum profit for providing their resources in VO. The administrators define VO policy and they are interested in the distributed environment overall performance optimization as well as in matching preferences of users and resource owners. A fair resource sharing implies that the interests of VO shareholders are met.

Every user job is submitted with a resource request – a list of requirements for the resources needed for a particular application execution. One of the most important parameters is a resource reservation time, during which the allocated nodes are utilized by the user job. For the overall job-flow execution optimization and a resource occupation time prediction, existing schedulers rely on the time specified in the job resource request. However, the reservation time is usually based on user inaccurate runtime estimates [14, 26]. In case, when the application is completed before the term specified in the resource request, the allocated resources remain underutilized. Moreover, if the job runtime estimation substantially differs from the real runtime, the job schedule may become ineffective in terms of optimization criteria defined in VO.

Thus, we outline two main job-flow optimization directions in the distributed computing environment. 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. The cyclic scheduling scheme (CSS) [27] 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 [28] may be related to this type of scheduling. Existing scheduling approaches are discussed in the next subsection.

2.2 Related Works

There are several resource selection and scheduling algorithms for parallel jobs and tasks with dependencies in distributed environments [17, 22, 23, 24, 29, 30, 31, 32, 33, 34, 35]. The scheduling problem in grid is NP-hard due to its combinatorial nature and many heuristic-based solutions have been proposed. In [17], heuristic algorithms for slot selection based on user-defined utility functions are introduced. NWIRE system [17] performs a slot window allocation based on the user defined efficiency criterion under the maximum total execution cost constraint. However, the optimization occurs only on the stage of the best found offer selection.

The paper [30] presents architecture and an algorithm for performing grid resources co-allocation without the need for advance reservations based on synchronous queuing (SQ) of subtasks. The objective of SQ is to minimize the co-allocation skew of all tasks requiring co-allocation. It enables SQ to over subscribe the re-

sources and hence to improve resource utilization. Mean utilization value is a single criterion in this model. However, advance reservation is effective to improve the co-allocation QoS. Moreover job control and resource management may be efficient using strategies. This means a combination of different algorithms and scheduling heuristics [3, 17, 22, 23, 24, 27, 29, 32, 34, 35] with consideration for multiple factors and criteria: the policy of resource allocation and administration, dynamical composition and heterogeneity of CPU nodes, tasks dependencies, etc. By combining the optimization criteria, VO administrators and users can form alternative search strategies for every job in the batch [27, 29]. Users may be interested in their jobs total execution cost minimizing or, for example, in the earliest possible jobs finish time, and in being able to affect the set of alternatives found by specifying the job distribution criteria. VO administrators in turn are interested in finding extreme alternatives characteristics values (e.g., total cost, total execution time) to form more flexible and, possibly, more effective combination of alternatives representing a batch execution schedule.

Advance reservation-based co-allocation algorithms are proposed in [22, 23, 24, 31, 32]. First fit resource selection algorithms (backtrack [22, 23] and NorduGrid [24] approaches) assign any job to the first set of slots matching the resource request conditions without any optimization. The co-allocation algorithms described in [31, 32, 33] suppose an exhaustive search and some of them are based on a linear integer programming (IP) [3, 32] or mixed-integer programming (MIP) model [33]. In [31] an online algorithm for co-allocating resources that provides support for advance reservations is proposed. The overall complexity of the algorithm for a successful scheduling attempt for the temporal space including a set of Q slots is $O\left(n_r \times Q \times (\log M)^2\right)$, where M is the number of servers in a computing system, and n_r is the reservation spatial size, i.e., the number of servers required for the given job. The co-allocation algorithm presented in [32] uses the 0-1 IP model with the goal of creating reservation plans satisfying user resource requirements. Users can specify a time frame for each resource: the earliest start time, the latest start time, and the job duration, where the user wants to reserve a time slot. This condition imposes restrictions for slots search only within this time frame. Moreover, the important factor is a complexity and an actual calculation time of the algorithm under consideration [32] especially with the assumption of the repeated use during the scheduling interval. The number of variables in the proposed algorithm becomes R^3 depending on the number of computer sites R. Thus, this approach may be inadequate for an on-line service in practical use. A linear IP-driven algorithm is proposed in [3]. It combines the capabilities of IP and genetic algorithm and allows to obtain the best metaschedule that minimizes the combined cost of all independent users in a coordinated manner. In [33], 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.

Backfilling [28] is a FCFS (first come – first served) method modification. In contrast to FCFS, backfilling requires user's jobs runtime estimates in order to re-

serve resources in advance. The resources are assigned to the jobs in a priority order, and the jobs are allocated to suitable resources if they are not already reserved for higher priority jobs. The advance reservation mechanism in backfilling guarantees to get the resources for higher priority jobs and allows the job queue order violation, which contributes to a higher overall resource utilization. The queue order violation occurs during the backfill stage when low priority jobs are attempted to be allocated to unreserved resources. With backfilling conservative variation a low priority job may be executed out of order, if it will not delay the execution of all higher priority jobs. Aggressive backfilling variation allows jobs to be executed out of the order only in case, when they do not delay the highest priority job execution.

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 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 [35] and to avoid resources fragmentation (see Section 2.3).

The scheduling techniques proposed in [3, 31, 32, 33, 34, 35] are efficient compared with other scheduling techniques under given criteria: the minimum processing cost, the overall makespan, resources utilization, load balancing, 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 user 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 [25, 27, 29]. Besides, in our approach the job start time and the finish time for slot search algorithms may be considered as criteria specified by users in accordance with the job total allocation cost. It offers the opportunity to perform more flexible scheduling solutions.

2.3 Cyclic Scheduling Scheme

Cyclic scheduling was proposed for a model based on a hierarchical job-flow management scheme [27]. Job-flow scheduling is performed in cycles by separate job batches on the basis of dynamically updated local schedules of computational nodes (Figure 1). Sets of available slots and their costs (C_j in Figure 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 (alternative offers for each batch job) 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 most efficient or optimal in terms of

the whole job batch execution in the current scheduling cycle is chosen (according to the VO policy).

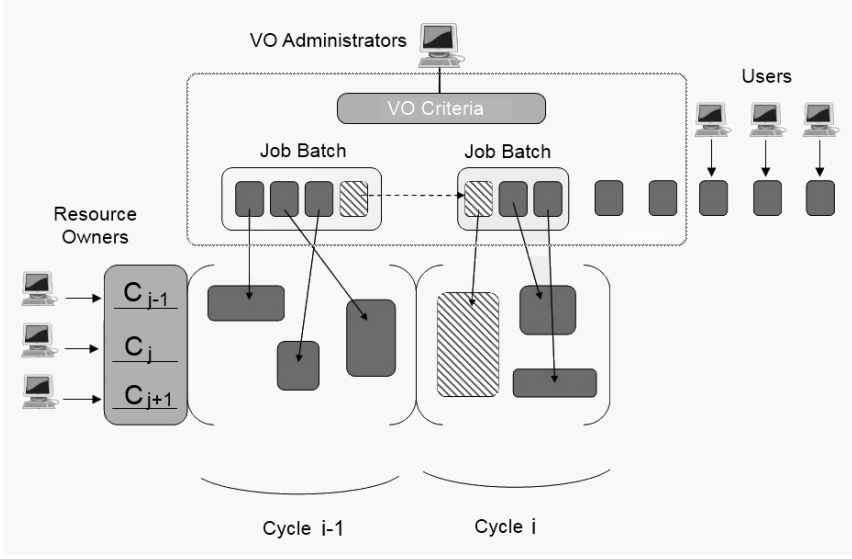


Figure 1. Job flow cyclic scheduling

Let S_i be the family of appropriate sets of slots for executing job i, $i = 1, \ldots, n$, 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 $t_i(s_j), j = 1, \ldots, N, N = |\bigcup_{i=1}^{n} S_i|$. Denote by S the family of appropriate sets of slots and by $\overline{s} = (s_1, \ldots, s_n), \overline{s} \in S$, 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 $g_i(s_j) \le g_i \le g^*$, where g_i is the level of the total expenses for the execution of a part of jobs from the batch (for example, jobs $i, i+1, \ldots, n$ or $i, i-1, \ldots, 1$) 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 $\overline{s} = (s_1, \ldots, s_n)$ is as follows:

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

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 $s_j \in S_i, i = \overline{1,n}, j \in \{1,\dots,N\}$ based on backward recursion are

$$\begin{aligned} f_i(g_i) &= \operatorname*{extr}_{s_j \in S_i}\{f_i(s_j) + f_{i+1}(g_i - g_i(s_j))\}, \qquad (2)\\ g_i(s_j) \le g_i \le g^*, \quad g^* &= \sum_{i=1}^{n} g_i^0(s_j), \quad i = \overline{1,n}\\ f_{n+1}(g_{n+1}) \equiv 0, \quad g_i &= g_{i-1}(s_k), \quad 1 < i \le n, \quad g_1 = g^*, \quad s_k \in S_{i-1}, \end{aligned}$$

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

The optimal expenses are determined from the equation

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

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

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

Here (4) represents the solution of the problem (1). An example of a resource expense level function in (1) is $t_i^0(s_j) = \left[\sum_{s_j} t_i(s_j)/l_i\right]$, where l_i is the number of admissible (alternative) sets of slots for the execution of job i, [.] 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

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

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

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

   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:

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

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

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

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:

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

The above functional Equations (6)–(9) are concretizations of (2) and are implemented as simulation environment components [27].

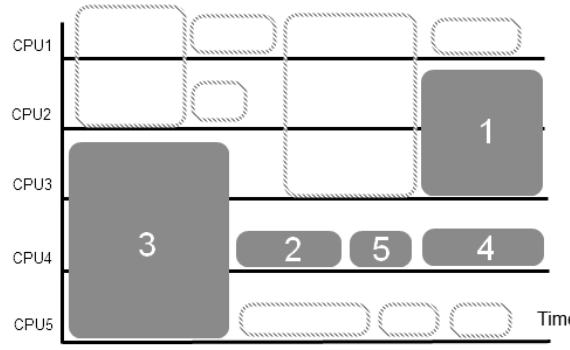


Figure 2. An example of alternatives allocation for a batch of five jobs

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 of the optimal combination (4) represents only the interests of VO upon the whole. In our previous work [25], two algorithms for slot selection AMP and ALP that feature linear complexity $O(p)$, where p is the number of available time-slots, were proposed. Both algorithms perform the search of the first fitting window without any optimization. AMP (Algorithm based on Maximal job Price), performing slot selection based on the maximum slot window cost, proved the advantage over ALP (Algorithm based on Local Price of slots) when applied to the above mentioned scheduling scheme. However, in order to accommodate an end user's job execution requirements, there is a need for more precise slot selection algorithms to consider various user demands along with the VO resource

management policy. 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.

Figure 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 section is dedicated to the CSS generalization and further development.

3 THE MODEL OF SCHEDULING AND FAIR RESOURCE SHARING

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. Batch-slicer is a generalization of CSS described above, and therefore it takes into account the interests of diverse VO stakeholders. In order to satisfy the user preferences, a desirable optimization criterion is introduced into the resource request format.

Unlike the so-called soft constraints [14] representing the user preferences, the optimization criterion defined in the resource request is considered during the stage of alternatives (slot sets) search. An Algorithm searching for Extreme Performance (AEP) described in details in [29] is used to select optimal alternatives under a given criterion. The job is a set of m interrelated tasks. The launch of any job requires a co-allocation of a specified number of slots, as well as in the classic backfilling variation [28]. The target is to scan a list of available slots and to select a window W of m parallel slots with a length of the required resource reservation time. The length of each slot in the window is determined by the performance rate of the resource on which it is allocated. The time length of an allocated window W is defined by the execution time of the task that is using the slowest CPU node, and in the case of heterogeneous resources, as a result one has a window with a "rough right

edge" (Figure 3). One can define a criterion on which the best matching window alternative is chosen. This can be a criterion crW for a minimum cost, a minimum execution runtime or, for example, a minimum energy consumption.

The scheme for an optimal window search by the specified criterion may be represented as follows:

Data: *slotList* – a list of available slots; *job* – a job for which the search is performed
Result: *bestWindow* – a window with the extreme criterion crW value
slotList = orderSystemSlotsByStartTime();
for ***each*** *slot* ***in*** *slotList* **do**
  if ***not****(properHardwareAndSoftware(slot.node))* **then**
    continue;
  end
  windowSlotList.add(slot);
  windowStartTime = slot.startTime;
  for ***each*** *wSlot* ***in*** *windowSlotList* **do**
    minLength = wSlot.node.getWorkingTimeEstimate();
    if *(wSlot.endTime – windowStartTime) < minLength* **then**
      windowSlotList.remove(wSlot);
    end
  end
  if *windowSlotList.size() ≥ job.nodesNeed* **then**
    curwindow = getBestWindow(windowSlotList);
    crW = getCriteriaValue(curWindow);
    if *crW > maxCriteriaValue* **then**
      maxCriteriaValue = crW;
      bestWindow = curWindow;
    end
  end
end

Algorithm 1: AEP-scheme for an optimal slot window selection

Finally, a variable *bestWindow* will contain an effective window by the given criterion crW.

The window search is performed on the list of all available slots sorted by their start time in ascending order (see Figure 3). This condition is necessary to examine every slot in the list and for operation of search algorithms of linear complexity [25, 27]. AEP can be compared to the algorithm of min/max value search in an array of flat values. The expanded window of size p "moves" through the ordered list of available slots. At each step any combination of m slots inside it (in the case, when $m \leq p$) can form a window that meets all the requirements to run the job. The effective on the specified criterion window of size m is selected from these p slots

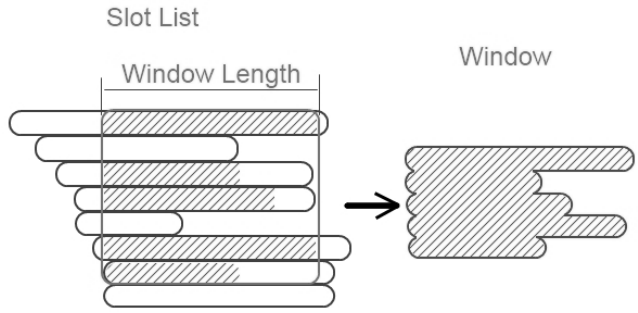


Figure 3. Window with a "rough right edge"

and compared with the results in the previous steps. By the end of the slot list the only solution with the best criterion crW value will be selected.

We considered as an example the problem of selecting a window of size m with a total cost no more than C from the list of $p > m$ slots (in the case, when $p = m$ the selection is trivial). The cost of using each of the slots according to their required time length is: $c_1, c_2, \ldots, c_p$. Each slot has a numeric characteristic z_q in accordance to crW. The total value of these characteristics should be minimized in the resulting window. Then the problem is formulated as follows: to minimize $a_1z_1 + a_2z_2 + \ldots + a_pz_p$, under $a_1c_1 + a_2c_2 + \ldots + a_pc_p \le C, \quad a_1 + a_2 + \ldots + a_p = m$, $a_q \in \{0, 1\}, q = 1, \ldots, p$.

Finding the coefficients $a_1, a_2, \ldots, a_p$ each of which takes integer values 0 or 1 (and the total number of "1" values is equal to m, because each job is the set of m interrelated tasks), determine the window with the specified criterion crW extreme value. Additional restrictions can be added, for example, considering the specified value of deadline. By combining the optimization criteria, VO administrators and users can form diverse slot selection and co-allocation strategies for every job in the batch.

Thus, a set of job execution alternatives is formed by the user preferences according to the individual application optimization criteria. At the same time the optimal alternatives combination choice is carried out in accordance with the criterion which implements VO policy. Resource owners receive an opportunity to manage their own profit and computational nodes utilization by varying local schedules and price establishing during the scheduling cycle.

Another difference between batch-slicer and CSS consists in the job system formation algorithm. Batch-slicer implies 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 [27], 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 (4), 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 (4) stage, and thereby increases the resource sharing efficiency according to VO policy. When separating the original batch to n sub-batches, where n is a total number of jobs in the batch (see Section 2.3), the algorithm will find the best sets of slots for each job according to the criteria specified in their resource requests. But in this case the efficiency of a whole job batch scheduling is not taken into account. On the other hand, when only a single sub-batch is "picked" from the original job batch the scheduling result will be identical to CSS application.

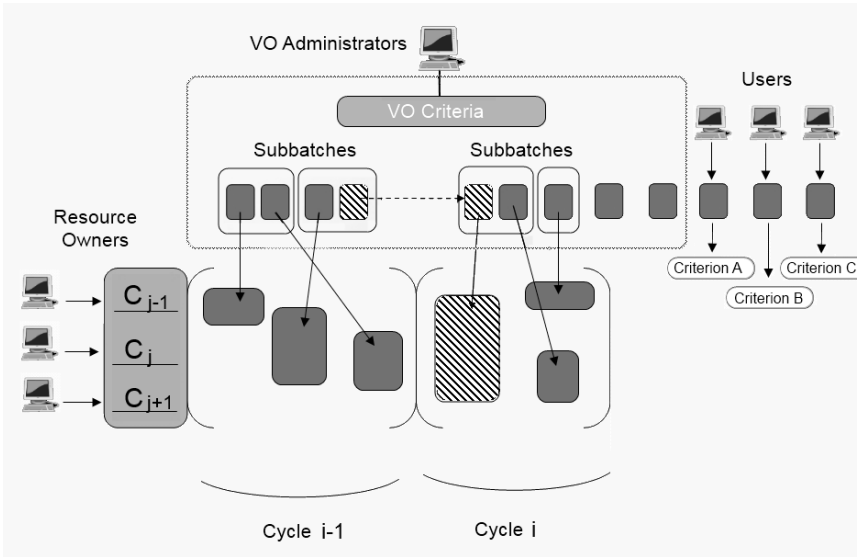


Figure 4. Job flow cyclic scheduling with batch-slicing

In view of described modifications, batch-slicer is schematically shown in Figure 4: an optimization criterion is specified for each job, and the job batch is separated to the sub-batches during the scheduling cycle.

4 CYCLIC SCHEDULING METHOD COMBINED WITH BACKFILLING

Each of the approaches described above has its advantages and disadvantages. Batch-slicer makes it possible to optimize the job-flow execution according to the VO shareholders preferences on condition that a sufficient number of alternatives was found for the batch jobs during the scheduling cycle. Backfilling 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.

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 complying 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) combines 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. The exact number of jobs to select into the first sub-batch to schedule with batch-slicer and the selection principle may depend on the related resource domain characteristics as well as on the job batch composition and general parameters.

5 SIMULATION ENVIRONMENT SETUP

A series of studies were carried out with the simulation environment [27] 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; resource requests and distributed computing environment generation; AEP slot processing; an algorithm for optimal alternatives combination selection; batch-slicer module; backfilling module; BSF module.

The main features of the simulation environment are as follows.

1. The job-flow and domain heterogeneous resources generation is performed in accordance with the random variables distribution functions with settings specified in the model for the real traces simulation.
2. Initial domain node utilization level is determined by the local tasks number and runtime. The initial CPU node utilization schedule is generated with the hypergeometric distribution.
3. The model supports different pricing mechanisms and the interaction between the VO stakeholders with economic principles.
4. The algorithms for job system formation, alternatives search and the best alternatives combination selection are implemented in the model.

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 $r \in [2, 10]$. Thus the resources with the highest performance level ($r = 10$) are generally able to execute jobs roughly twice as fast as medium performance level nodes ($r = 6$), while nodes with the lowest performance ($r = 2$) are three times slower. 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 [14]). 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. 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.

A special study is a simulation of a complete scheduling cycle for the same job batch independently by proposed and known algorithms. In each experiment, a job batch presented as a resource requests list is performed and then a resource environment composition with local utilization schedules is generated. Thus the study is based on scheduling results obtained with the same input (job batch) using different scheduling algorithms comparison.

6 CSS AND FAIR RESOURCE SHARING EXPERIMENTAL STUDIES

The goal of the investigation is to verify basic concepts of fair resource sharing, i.e. to prove that each VO member has a possibility to affect the process of scheduling according to his preferences (see Section 2).

6.1 AEP-Based Heuristic Strategies

The need to choose alternative sets of slots for every batch job increases the complexity of the whole scheduling scheme [27]. With a large number of available slots the search algorithm execution time may become inadequate. However, it is possible to mention some typical optimization problems based on the AEP scheme that can be solved with a relatively decreased complexity. These include problems of total job cost minimizing, total runtime minimizing, the window formation with the minimal start/finish time.

For the proposed AEP efficiency analysis the following heuristic strategies and their algorithmic implementations were added to the simulation model [27].

1. AMP – searching for slot windows with the earliest start time. This scheme was introduced in [25].
2. MinRunTime – this strategy performs a search for a single alternative with the minimum execution runtime. Given the nature of determining a window runtime, which is equal to the length of the longest composing slot, the following scheme may be proposed:

Data: *windowSlotList* – a list of slots in the window
Result: $resultWindowList$ – a list of n slots with the minimum runtime
orderSlotsByCostAscending(windowSlotList);
resultWindowList = getSubList(0,n, windowSlotList);
spareWindowList = getSubList(n,m, windowSlotList);
while *spareWindowList.size() > 0* **do**
  longSlot = getLongestSlot(resultWindowList);
  shortSlot = getCheapestSlot(spareWindowList);
  spareSlotList.remove(shortSlot);
  if *(shortSlot.length < longSlot.length)* ***and*** *((()resultWindowList.sumCost – longSlot.cost + shortSlot.cost) < job.costLimit)* **then**
    resultWindowList.remove(longSlot);
    resultWindowList.add(shortSlot);
  end
end

Algorithm 2: Simplified runtime minimization subwindow selection scheme

As a result, the suitable window of the minimum time length will be formed in a variable $resultWindow$. The algorithm described consists of the consecutive attempts to substitute the longest slot in the current window (the $resultWindow$ variable) with another shorter one that will not be too expensive. In case when it is impossible to substitute the slots without violating the constraint on the maximum window allocation cost, the current resultWindow configuration is declared to have the minimum runtime.

3. MinFinish – searching for alternatives with the earliest finish time. This strategy may be implemented using the runtime minimizing procedure presented above. Indeed, the expanded window has a start time $tStart$ equal to the start time of the last added suitable slot. The minimum finish time for a window on this set of slots is $(tStart + minRuntime)$, where $minRuntime$ is the minimum window length. The value of $minRuntime$ can be calculated similarly to the runtime minimizing procedure described above. Thus, by selecting a window with the earliest completion time at each step of the algorithm, the required window will be allocated in the end of the slot list.

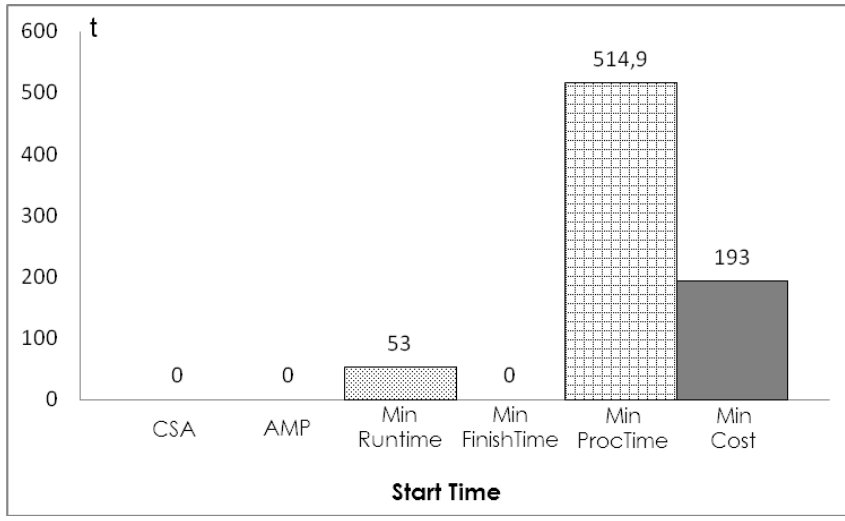
4. MinCost – searching for a single alternative with the minimum total allocation cost on the scheduling interval. For this purpose in the AEP search scheme m slots with the minimum sum cost should be chosen. If at each step of the algorithm a window with the minimum sum cost is selected, at the end the window with the best value of the criterion crW will be guaranteed to have overall minimum total allocation cost at the given scheduling interval.
5. MinProcTime – this strategy performs a search for a single alternative with the minimum total node execution time defined as a sum of the composing slots time lengths. It is worth mentioning that this implementation is simplified and does not guarantee an optimal result and only partially matches the AEP scheme, because a random window is selected.
6. Common Stats, AMP (further referred to as CSA) – the strategy for searching multiple alternatives using AMP. Similar to the general searching scheme [25], a set of suitable alternatives, disjointed by the slots, is allocated for each job. To compare the search results with the strategies 1-5, presented above, only alternatives with the extreme value of the given criterion will be selected, so the optimization will take place at the selection process. The criteria include the start time, the finish time, the total execution cost, the minimum runtime and the processor time used.

It is worth mentioning that all proposed AEP implementations have a linear complexity $O(p)$: algorithms "move" through the list of p available slots in the direction of non-decreasing start time without turning back or reviewing previous steps.

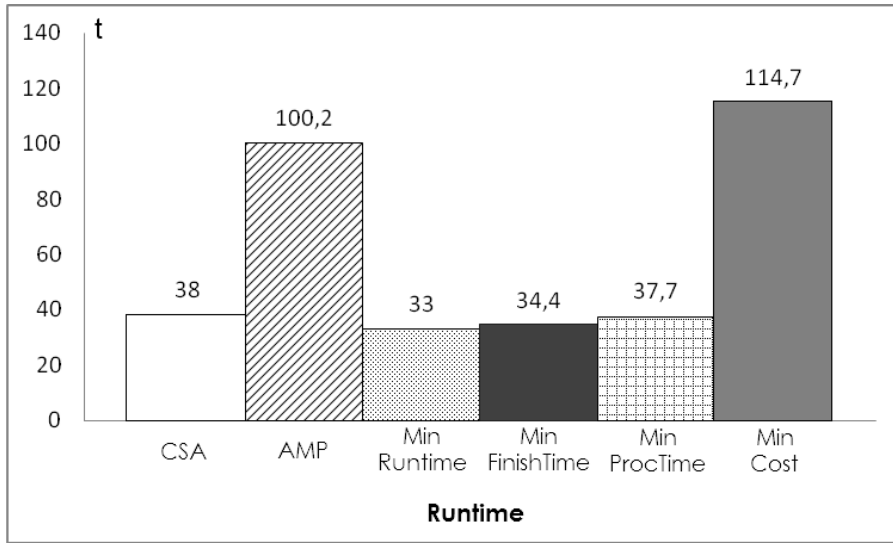
The goal of the experiment is to examine AEP implementations: to analyze co-allocation strategies with different efficiency criteria, to compare the results with AMP and to estimate the possibility of using in real systems considering the algorithm execution time for each strategy. In each experiment a generation of the distributed environment that consists of 100 CPU nodes was performed. The relatively high number of the generated nodes has been chosen to allow CSA to find more slot alternatives. Therefore more effective alternatives could be selected for the searching results comparison based on the given criteria. The level of the resource initial load with the local and high priority jobs at the scheduling interval $[0; 600]$ was generated by the hyper-geometric distribution in the range from 10 % to 50 % for each CPU node. Based on the generated environment the algorithms performed the search for a single initial job that required an allocation of 5 parallel slots for 150 units of time. The maximum total execution cost according to user requirements was set to 1 500. This value generally will not allow using the most expensive (and usually the most efficient) CPU nodes.

The results of the 5 000 simulated scheduling cycles are presented in Figure 5.

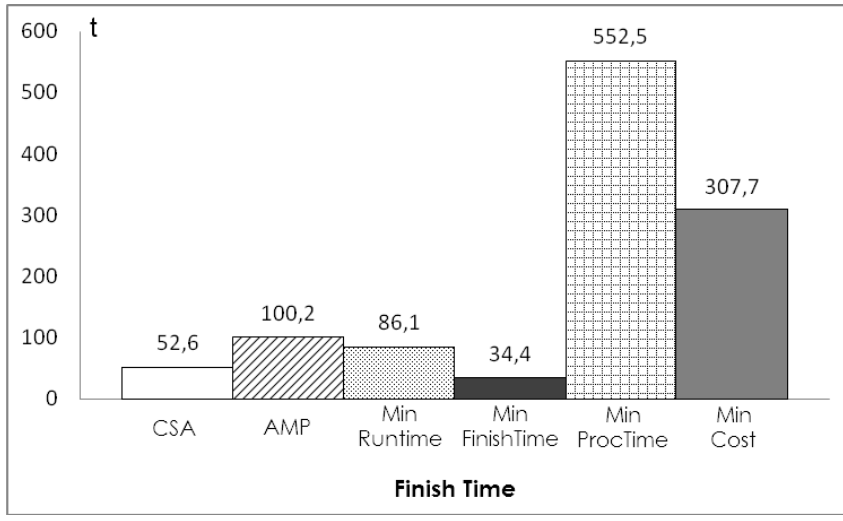
The obtained values of the total job execution cost are as follows: AMP – 1 445.2; minFinish – 1 464.2; minCost – 1 027.3; minRuntime – 1 464.9; minProcTime – 1 342.1; CSA – 1 352. Each full AEP-based strategy was able to obtain the best result in accordance with the given criterion: start time (Figure 5 a)); runtime


600
500
400
300
200
100
0
t
514,9
193
53
0
0
0
CSA
AMP
Min Runtime
Min FinishTime
Min ProcTime
Min Cost
Start Time

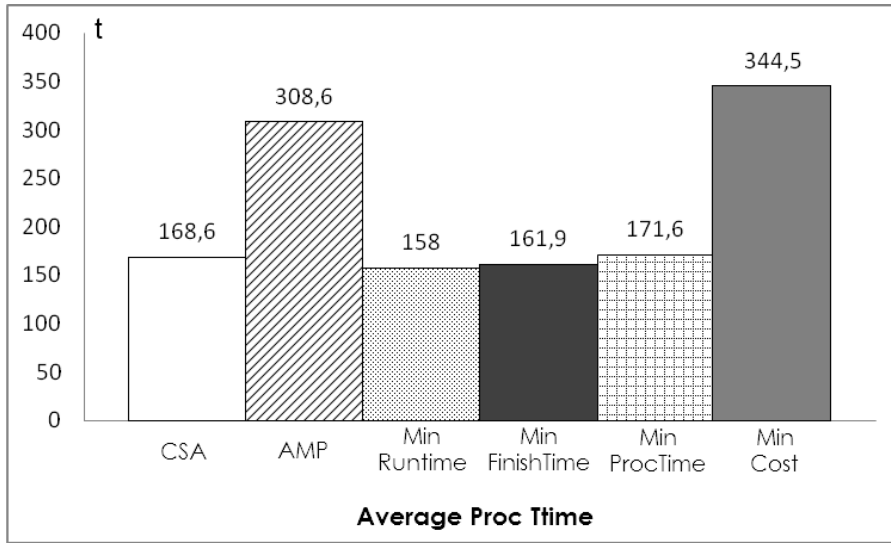
a)


140
120
100
80
60
40
20
0
t
114,7
100,2
38
33
34,4
37,7
CSA
AMP
Min Runtime
Min FinishTime
Min ProcTime
Min Cost
Runtime

b)



c)



d)

Figure 5. Average a) start time, b) runtime, c) finish time, and d) CPU usage time

(Figure 5 b)); finish time (Figure 5 c)); CPU usage time (Figure 5 d)). Besides, a single run of the AEP-like algorithm had an advantage of 10 %–50 % over suitable alternatives found by AMP with a respect to the specified criterion. According to the experimental results, on one hand, the best scheme with top results in start time, finish time, runtime and CPU usage time was minFinish. Though, in order to obtain such results, the algorithm spent almost all user specified budget (1 464 of 1 500). On the other hand, the minCost strategy was designed precisely to minimize execution expenses and provides 43 % advantage over minFinish (1 027 of 1 500), but the drawback is a more than modest results by other criteria considered.

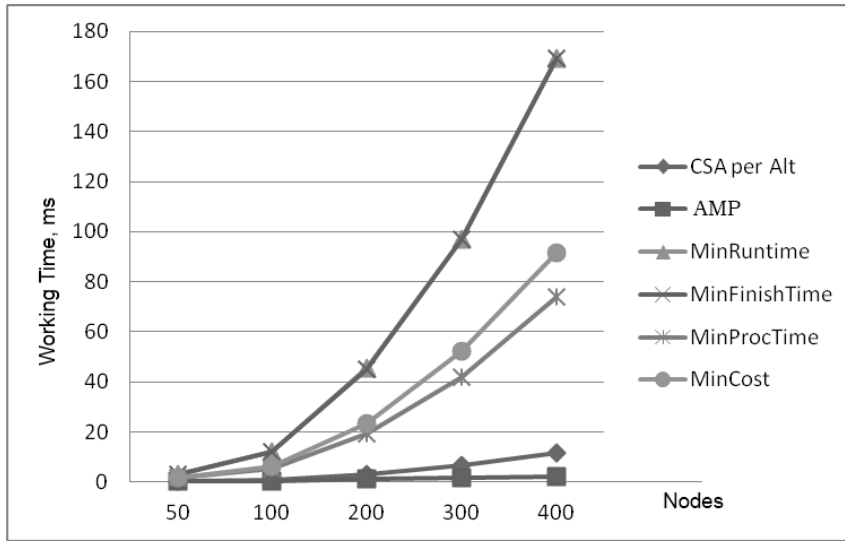
The important factor is a complexity and an actual working time of the algorithms implementing heuristic strategies. Figure 6 a) shows the actual algorithms execution time in miliseconds measured depending on the number of CPU nodes. The simulation was performed on a regular PC workstation with Intel Core i3 (2 cores @ 2.93 GHz), 3 GB RAM on JRE 1.6, and 1 000 separate experiments were simulated for each value of the processor nodes numbers $\{50, 100, 200, 300, 400\}$.

The CSA strategy has the longest working time that on the average almost reaches 3 seconds when 400 nodes are available. A curve "CSA per Alt" in Figure 6 a) represents an average working time for the CSA algorithm in recalculation for one alternative. AEP-based algorithms feature a quadratic complexity.

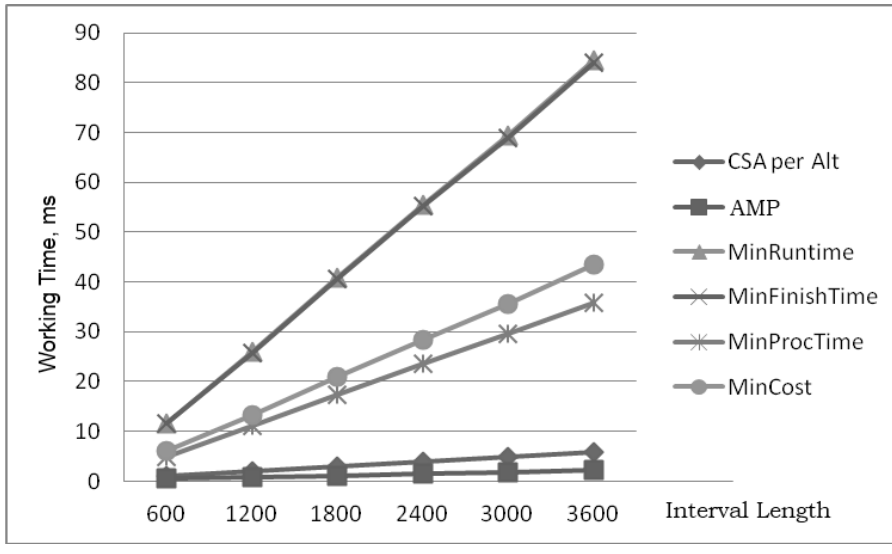
Figure 6 b) presents the algorithms working time in milliseconds measured depending on the scheduling interval length. Overall 1 000 single experiments were conducted for each value of the interval length $\{600, 1\,200, 1\,800, 2\,400, 3\,000, 3\,600\}$ and for each considered algorithm an average working time was obtained. The experiment simulation parameters and assumptions were the same as described earlier in this section, apart from the scheduling interval length. A number of CPU nodes was set to 100. Similarly to the previous experiment, CSA had the longest working time (about 2.5 seconds with the scheduling interval length equal to 3 600 model time units), which is mainly caused by the relatively large number of the formed execution alternatives (on the average more than 400 alternatives on the 3 600 interval length).

Analyzing the presented values it is easy to see that all proposed algorithms have a linear complexity with the respect to the length of the scheduling interval and, hence, to the number of the available slots. The minProcTime strategy stands apart and represents a class of simplified AEP implementations with a noticeably reduced working time. And though the scheme compared to other considered algorithms, did not provide any remarkable results, it was on the average only 2 % less effective than the CSA scheme by the dedicated CPU usage criterion (see Figure 5 d)). At the same time its reduced complexity and actual working time (see Figure 6 b)) allow to use it in a large wide scale distributed systems when other optimization search algorithms prove to be too slow.

As a result it may be stated that the advantage of AEP-based strategies over the general CSA scheme was shown for each of the considered criteria: start time, finish time, runtime, CPU usage time, and total cost.



a)



b)

Figure 6. Average a) working time duration depending on the available CPU nodes number and b) the scheduling interval length

6.2 Taking into Account VO Users' Preferences in Batch-Slicer

Taking into account VO users' preferences, the analysis of scheduling results for individual jobs is carried out by comparing the proposed batch-slicer approach with the initial CSS. The latter does not perform optimization during the stage of alternatives selection. Thus, there are two approaches considered in the experiment:

1. scheduling of the job batch with batch-slicer where alternatives search is performed based on AEP taking into account the criterion specified in resource request,
2. scheduling of the job batch with CSS where alternatives search is performed by choosing the first fit alternative.

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 and cost (AEP minimizes the value of the specified criterion).

Criterion	N_A	Start time	Execution time	Finish Time	Cost
Start time	12.8	171.7	56.1	227.8	1 281.1
Execution time	10.6	214.5	**39.3**	253.9	1 278.5
Finish time	12.2	**169.6**	45	**205.5**	1 283.2
Cost	12.9	262.6	55.5	318	**1 098.3**
CSS	12.1	222	50.3	272.3	1 248.4

Table 1. Scheduling results with VO users' preferences

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 job batch with initial CSS without optimization at the stage of alternatives search. Simulation of 5 000 individual scheduling cycles was conducted. As can be seen from Table 1, 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 MinFinish 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 %. 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. Average number of jobs per each group having the same optimization criterion equals 5 on average. This fits the use of uniform distribution when choosing one of the four optimization criteria for each of the 20 batch jobs.

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 the resources are allocated among the jobs uniformly.

6.3 Optimization of Job Batch Execution in VO

The next experiment is dedicated to comparing the scheduling results when slicing the initial job batch in batch-slicer into different number of sub-batches and at different levels of environment utilization. The experiment allows estimating the efficiency of scheduling in different modes with different input data. Modes comparison was performed on the basis of job batch allocation results on full scheduling cycle including initial environment generation, composition of batches and sub-batches and then their sequential scheduling.

When choosing the optimal execution alternatives combination the average job execution time T_{CPU} minimization task 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 7 shows the value of T_{CPU} depending on the number of sub-batches $k \in \{1, 2, 3, 5, 6, 10, 20\}$ into which the initial batch is sliced and the level of environment utilization. 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. The experiments were conducted under high ($Y = 0.3$), medium ($Y = 0.03$) and low utilization levels ($Y < 0.0002$). Thus the number of failures in the conducted series of experiments differs at the minimum by the order of magnitude of one.

As a result of the job batch scheduling experiment the following patterns were revealed. 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 a relative number of failures Y. When increasing the level of available resources the number of alternatives for the individual job execution increases, the relative number of failures Y decreases, and the total cost of job batch execution decreases. The experiment results show that slicing of the initial batch into sub-batches and their sequential independent scheduling allows to increase the number of execution alternatives for the batch jobs and to perform more efficient execution schedules. So, at a high execution environment the utilization rate, the best efficiency and the least number of failures are provided by slicing into fewer sub-batches. On the other hand, at a low utilization of available resources it is advantageous to slice into a higher number of sub-batches up to scheduling the jobs individually (see Figure 7). Another advantage of batch-slicer in comparison with CSS is decreasing of total execution cost as the level of resource utilization becomes lower. CSS tries to use the entire admissible budget b^* for the job batch execution

by choosing the corresponding set of alternatives. At the same time, when scheduling sub-batches with a small number of jobs, the choice is often confined to a few alternatives whose cost is not necessarily close to the admissible budget limitation.

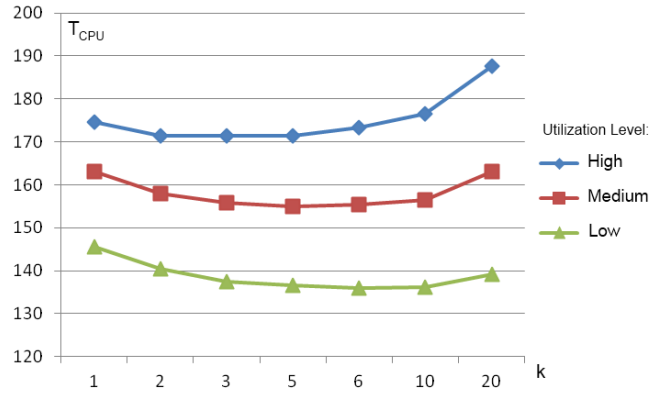


Figure 7. Average jobs execution time T_{CPU} depending on the number of sub-batches k

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 it can also provide a better value of the target criterion in comparison with CSS by slicing into sub-batches (the least value of the target criterion – job batch total execution time – was achieved when slicing the job batch into 5 sub-batches with four jobs in each of them).

6.4 Taking into Account VO Resource Owners' Preferences

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 – relative resource utilization average value in the scheduling interval; P – average profit made by the resource owner, and Y – relative number of scheduling failures.

c	L_c	U	P	Y
2	256.6	0.44	527.1	0
4	234.9	0.39	939.6	0.001
6	185.4	0.31	1 112.3	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

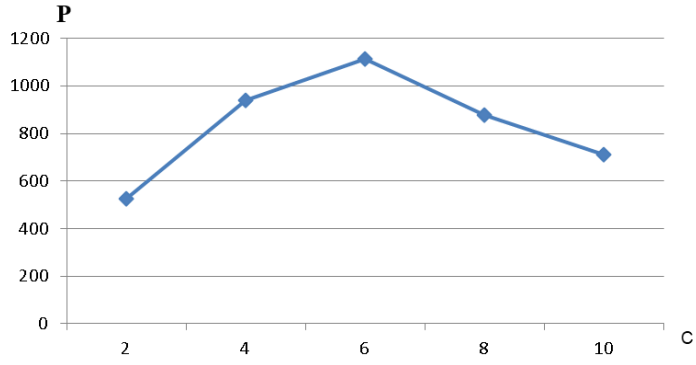


Figure 8. A resource owner's profit P depending on the proposed price c

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. The profit value received by a resource owner for providing a single computational node is illustrated in Figure 8.

7 EXPERIMENTAL STUDIES OF RESOURCE USE EFFICIENCY IN THE CYCLIC SCHEME

7.1 Studies of Combined Scheduling Method BSF

The efficiency of scheduling with BSF combined approach can be considered from two viewpoints at the same time: on the one hand, from the viewpoint of criterion value optimization in the specific VO, job batch total execution time (7) for instance, and, on the other hand, from the environment utilization level and the batch job start time minimization viewpoint.

Figure 9 shows the batch job average execution time T_{CPU} and the average start time T_{start} depending on the ratio according to which slicing into sub-batches was made: n_{CSS} – the number of jobs in the first sub-batch, scheduled with CSS, n – total number of jobs in the batch.

Slicing into sub-batches was made based on a priority – the order of jobs in the batch – without taking into account the characteristics of the jobs themselves. The scheduling results presented in Figure 9 are obtained based on simulation of 5 000 independent scheduling cycles at a medium level of resource utilization according to the settings described in Section 5. As seen from Figure 9, if a major part of the job is scheduled with batch-slicer then a better value of the target VO scheduling criterion – the execution time T_{CPU} is achieved, but the average job start time

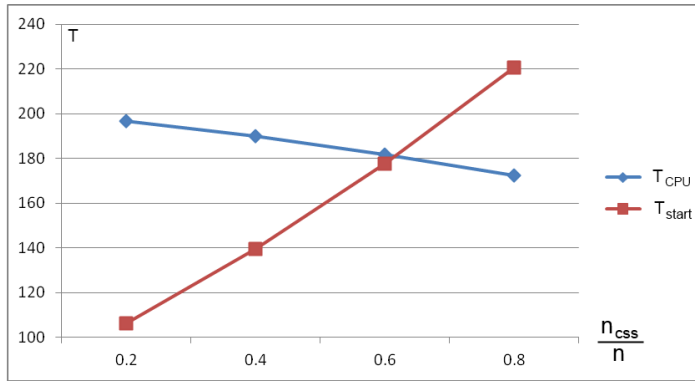


Figure 9. Average job execution T_{CPU} and start T_{start} time in BSF

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 given to the cross point of graphs in Figure 9. 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. So, BSF shows "compromise" values of the discussed characteristics compared to BS and backfilling.

7.2 Experimental Studies of Consistency of Schedules Based on Job Execution Time Estimate

Let us consider scheduling efficiency study results and consistency of schedules performed with batch-slicer and CSS and based on the job execution time estimate which is specified in the resource request. 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. Early job completion, untimely resource release and idleness may negatively affect the efficiency of the job batch execution against the criteria specified by VO stakeholders and to make the schedule inconsistent. 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 an 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 the real job execution time differed considerably from the 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. Thus, at worst, the real execution time could differ from user estimate by 5 times.

Table 3 contains the average job execution time values (the target optimization criterion) and 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).

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

It can be seen from Table 3 that even if the difference between the resource reservation time and the real job execution time is significant the advantage of batch-slicer over backfilling against the VO target optimization criterion not only remains but even increases. That is because the backfilling does not optimize against the criteria different from a start time and a more compact job location (real start time of jobs scheduled with backfilling is reduced by 46 % on average) uses almost all the available resources including those less advantageous against the target criterion.

Thus, results of this stress test show that preliminary schedules formed in the beginning of the scheduling cycle are consistent against the criteria determined in VO in the case when the real execution time differs significantly from the resource reservation time. Note that an additional advantage can be achieved by rescheduling when taking into account the information about current utilization of computational nodes.

8 CONCLUSIONS AND FUTURE WORK

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, we take into consideration the economic principles. We proposed a solution of the problem in a fair resource sharing among VO stakeholders.

The advantage over initial CSS when scheduling the job flow reaches 7 % and in the terms of a single job execution it reaches 25 % at a medium level of environment utilization. Resource owners can vary the resource provision unit cost (by offering discounts, for instance) to maximize the profit or to achieve the necessary resource utilization level. Based on the union of CSS and backfilling, a combined approach BSF is proposed. The approach shows compromise results compared to batch-slicer and backfilling, namely it allows utilizing the available resources efficiently

(by means of backfilling) when efficiently executing a part of jobs in VO (by means of optimization in batch-slicer). The consistency of scheduling made with batch-slicer when real job execution time is significantly different from user's estimate is shown.

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 information about computational nodes current utilization.

Acknowledgements

This work was partially supported by the Council on Grants of the President of the Russian Federation for State Support of Young Scientists and Leading Scientific Schools (grants YPhD-4148.2015.9 and SS-362.2014.9), RFBR (grants 15-07-02259 and 15-07-03401), the Ministry on Education and Science of the Russian Federation, task No. 2014/123 (project No. 2268), and by the Russian Science Foundation (project No. 15-11-10010).

REFERENCES

[1] Garg, S. K.—Buyya, R.—Siegel, H. J.: Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-Off Management. 32nd Australasian Computer Science Conference (ACSC 2009), Wellington, New Zealand, 2009, pp. 151–159.

[2] Degabriele, J. P.—Pym, D.: Economic Aspects of a Utility Computing Service. Technical Report HPL-2007-101, Trusted Systems Laboratory, HP Laboratories Bristol, 2007, pp. 1–23.

[3] Garg, S. K.—Yeo, C. S.—Anandasivam, A.—Buyya, R.: Environment-Conscious Scheduling of HPC Applications on Distributed Cloud-Oriented Data Centers. J. Parallel and Distributed Computing, Vol. 71, 2011, No. 6, pp. 732–749.

[4] Tesauro, G.—Bredin, J. L.: Strategic Sequential Bidding in Auctions Using Dynamic Programming. 1st International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, New York 2002, Part 2, pp. 591–598.

[5] Thain, D.—Tannenbaum, T.—Livny, M.: Distributed Computing in Practice: The Condor Experience. J. Concurrency and Computation: Practice and Experience, Vol. 17, 2004, No. 2-4, pp. 323–356.

[6] Berman, F.: High-Performance Schedulers. In: Foster, I., Kesselman, C. (Eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco 1999, pp. 279–309.

[7] Yang, Y.—Raadt, K.—Casanova, H.: Multiround Algorithms for Scheduling Divisible Loads. IEEE Trans. Parallel and Distributed Systems, Vol. 16, 2005, No. 8, pp. 1092–1102.

[8] Natrajan, A.—Humphrey, M. A.—Grimshaw, A. S.: Grid Resource Management in Legion. In: Nabrzyski, J., Schopf, J. M., Weglarz J. (Eds.): Grid Resource

Management. State of the Art and Future Trends. Kluwer Academic Publishers, Boston 2000, pp. 145–160.

[9] Beiriger, J.—Johnson, W.—Bivens, H.: Constructing the ASCI Grid. 9th IEEE Symposium on High Performance Distributed Computing, IEEE Press, New York 2000, pp. 193–200.

[10] Frey, J.—Foster, I.—Livny, M.: Condor-G: A Computation Management Agent for Multi-Institutional Grids. 10th International Symposium on High-Performance Distributed Computing, IEEE Press, New York 2001, pp. 55–66.

[11] Abramson, D.—Giddy J.—Kotler L.: High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? International Parallel and Distributed Processing Symposium, IEEE Press, New York 2000, pp. 520–528.

[12] Foster, I.—Kesselman C.—Tuecke S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. of High Performance Computing Applications, Vol. 15, 2001, No. 3, pp. 200–222.

[13] Ranganathan, K.—Foster, I.: Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. 11th IEEE International Symposium on High Performance Distributed Computing, IEEE Press, New York 2002, pp. 376–381.

[14] Kurowski, K.—Nabrzyski, J.—Oleksiak, A.—Weglarz, J.: Multicriteria Aspects of Grid Resource Management. In: Nabrzyski, J., Schopf, J. M., Weglarz, J. (Eds.): Grid Resource Management. State of the art and Future Trends. Kluwer Academic Publishers, Boston 2003, pp. 271–293.

[15] Garg S. K.—Konugurthi P.—Buyya R.: A Linear Programming-Driven Genetic Algorithm for Meta-Scheduling on Utility Grids. J. Par., Emergent and Distr. Systems, Vol. 26, 2011, pp. 493–517.

[16] Buyya, R.—Abramson, D.—Giddy, J.: Economic Models for Resource Management and Scheduling in Grid Computing. J. Concurrency and Computation, Vol. 14, 2002, No. 5, pp. 1507–1542.

[17] Ernemann, C.—Hamscher, V.—Yahyapour, R.: Economic Scheduling in Grid Computing. In: Feitelson, D. G., Rudolph, L., Schwiegelshohn, U. (Eds.): JSSPP 2002, Springer, Heidelberg, LNCS, Vol. 2537, 2002, pp. 128–152.

[18] Lee, Y. C.—Wang, C.—Zomaya, A. Y.—Zhou, B. B.: Profit-Driven Scheduling for Cloud Services with Data Access Awareness. J. Par. and Distr. Computing, Vol. 72, 2012, No. 4, pp. 591–602.

[19] Toporkov, V. V.: Job and Application-Level Scheduling in Distributed Computing. Special Issue on ICIT 2009 Conference – Applied Computing, Ubiquitous Computing and Communication J., Vol. 4, 2009, No. 3, pp. 559–570.

[20] Toporkov, V. V.—Toporkova, A.—Tselishchev, A.—Yemelyanov, D.: Job and Application-Level Scheduling: An Integrated Approach for Achieving Quality of Service in Distributed Computing. 4th International Conference on Dependability of Computer Systems, IEEE CS Press, Los Alamitos 2009, pp. 202–209.

[21] Toporkov, V.: Application-Level and Job-Flow Scheduling: An Approach for Achieving Quality of Service in Distributed Computing. In: Malyshkin, V. (Ed.): PaCT, 2009, Springer, Heidelberg, LNCS, Vol. 5968, 2009, pp. 350–359.

[22] Aida, K.—Casanova, H.: Scheduling Mixed-Parallel Applications with Advance Reservations. 17th IEEE Int. Symposium on HPDC, IEEE CS Press, New York 2008, pp. 65–74.

[23] Ando, S.—Aida, K.: Evaluation of Scheduling Algorithms for Advance Reservations. Information Processing Society of Japan SIG Notes. HPC-113, 2007, pp. 37–42.

[24] Elmroth, E.—Tordsson, J.: A Standards-Based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation and Cross-Grid Interoperability. J. of Concurrency and Computation, Vol. 25, 2009, No. 18, pp. 2298–2335.

[25] Toporkov, V.—Toporkova, A.—Bobchenkov, A.—Yemelyanov, D.: Resource Selection Algorithms for Economic Scheduling in Distributed Systems. Procedia Computer Science, Elsevier 4, 2011, pp. 2267–2276.

[26] Lee, S. B.—Schwartzman, Y.—Hardy, J.—Snavely, A.: Are User Runtime Estimates Inherently Inaccurate? In: Feitelson, D. G., Rudolph, L., Schwiegelshohn, U. (Eds.): JSSPP 2004, Springer, Heidelberg, LNCS, Vol. 3277, 2004, pp. 253–263.

[27] Toporkov, V.—Tselishchev, A.—Yemelyanov, D.—Bobchenkov, A.: Dependable Strategies for Job-Flows Dispatching and Scheduling in Virtual Organizations of Distributed Computing Environments. Complex Systems and Dependability, AISC, Springer, Heidelberg, Vol. 170, 2012, pp. 240–255.

[28] Moab Adaptive Computing Suite, `http://www.adaptivecomputing.com/products/moab-adaptive-computing-suite.php`.

[29] Toporkov, V.—Toporkova, A.—Tselishchev, A.—Yemelyanov, D.: Heuristic Co-Allocation Strategies in Distributed Computing with Non-Dedicated Resources. In: Zavoral, F., Jung, J. J., Badica, C. (Eds.): 7th International Symposium on Intelligent Distributed Computing. Studies in Computational Intelligence, Springer, Heidelberg, Vol. 511, 2013, pp. 109–118.

[30] Azzedin, F.—Maheswaran, M.—Arnason, N.: A Synchronous Co-Allocation Mechanism for Grid Computing Systems. Cluster Computing, Vol. 7, 2004, pp. 39–49.

[31] Castillo, C.—Rouskas, G. N.—Harfoush, K.: Resource Co-Allocation for Large-Scale Distributed Environments. 18th ACM International Symposium on High Performance Distributed Computing, ACM, New York 2009, pp. 137–150.

[32] Takefusa, A.—Nakada, H.—Kudoh, T.—Tanaka, Y.: An Advance Reservation-Based Co-Allocation Algorithm for Distributed Computers and Network Bandwidth on QoS-Guaranteed Grids. In: Frachtenberg, E., Schwiegelshohn, U. (Eds.): JSSPP 2010, Springer, Heidelberg, LNCS, Vol. 6253, 2010, pp. 16–34.

[33] Blanco, H.—Guirado, F.—Lérida, J. L.—Albornoz, V. M.: MIP Model Scheduling for Multi-Clusters. Euro-Par 2012, Springer, Heidelberg, LNCS, Vol. 7640, 2012, pp. 196–206.

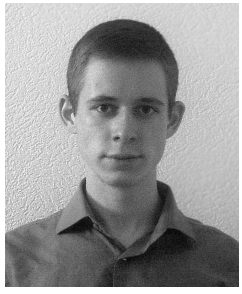
[34] Moise, D.—Moise, I.—Pop, F.—Cristea, V.: Resource Co-Allocation for Scheduling Tasks with Dependencies in Grid. The Second International Workshop on High Performance in Grid Middleware (HiPerGRID 2008), Bucharest, Romania, IEEE Romania, 2008, pp. 41–48.

[35] Olteanu, A.—Pop, F.—Dobre, C.—Cristea, V.: A Dynamic Rescheduling Algorithm for Resource Management in Large Scale Dependable Distributed

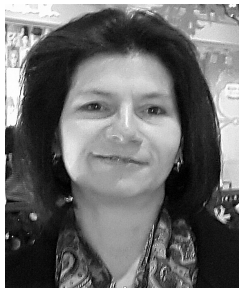
Systems. Computers and Mathematics with Applications, Vol. 63, 2012, No. 9, pp. 1409–1423.

Victor Toporkov received his D.Sc. degree in computer science from Moscow Power Engineering Institute (MPEI) in 2000. Currently, he is a Head of the Computer Science Department at National Research University MPEI, where he is Full Professor. His primary research interests focus on distributed computing, resource management and scheduling in grid. He is the author and co-author of about 200 papers in computer and computational sciences.

Dmitry Yemelyanov received his B.Sc. (2008), M.Sc. (2010) and Ph.D. (2013) degrees in computer science from National Research University MPEI, where he currently works as Assistant Professor at the Computer Science Department. He is experienced in programming, telecommunications and networks. His research topics include distributed computing, scheduling and resource management.



Petr Potekhin received his M.Sc. degree in computer science from National Research University MPEI. Currently he is a Ph.D. student at the Computer Science Department, National Research University MPEI. He has experience in software tools development and integration. His research interests include distributed computing, scheduling problems in grid, and job framework generation for efficient scheduling.



Anna Toporkova received her Ph.D. degree in computer science from Moscow State Institute of Electronics and Mathematics in 2001. She is now Associate Professor at the Computer Engineering Department, National Research University Higher School of Economics (Moscow). Her research interests include intelligent distributed computing, scheduling and optimization techniques.

Alexey Tselishchev received his Dipl.-Ing. degree in engineering informatics from Technical University Ilmenau (Germany) in 2007, M.Sc. degree in computer science in 2008, and Ph.D. degree in computer science in 2011 from National Research University MPEI. His research interests include distributed computing algorithms and computer security. He has published about 50 papers working as a research fellow at European Organization for Nuclear Research, National Research University MPEI, and as a security and identity management architect.