Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments

Klavdiya Bochenina, Nikolay Butakov, Alexander Boukhanovsky

PII: S0167-739X(15)00262-9
DOI: http://dx.doi.org/10.1016/j.future.2015.08.009
Reference: FUTURE 2821

To appear in: *Future Generation Computer Systems*

Received date: 23 April 2015
Revised date: 3 August 2015
Accepted date: 18 August 2015

Highlights

- We present an approach to scheduling of multiple workflows with soft deadlines;
- We analyze the impact of time restrictions on a quality of schedules;
- We perform experiments with several types of synthetic and domain-specific data sets;
- Accounting of time restrictions leads to the increase of the quality of schedules;
- Clustering-based scheduling scheme outperforms task-based and workflow-based schemes.

Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments

Klavdiya Bochenina[a], Nikolay Butakov[b], Alexander Boukhanovsky[c]
e-Science Research Institute, ITMO University
49, Kronverksky Pr., St. Petersburg, 197101, Russia
E-mail: k.bochenina@gmail.com, alipoov.nb@gmail.com, avb_mail@mail.ru

Abstract

Typical patterns of using scientific workflows include their periodical executions using a fixed set of computational resources. Using the statistics from multiple runs, one can accurately estimate task execution and communication times to apply static scheduling algorithms. Several workflows with known estimates could be combined into a set to improve the resulting schedule. In this paper, we consider the mapping of multiple workflows to partially available heterogeneous resources. The problem is how to fill free time windows with tasks from different workflows, taking into account users' requirements of the urgency of the results of calculations. To estimate quality of schedules for several workflows with various soft deadlines, we introduce the unified metric incorporating levels of meeting constraints and fairness of resource distribution.

The main goal of the work was to develop a set of algorithms implementing different scheduling strategies for multiple workflows with soft deadlines in a non-dedicated environment, and to perform a comparative analysis of these strategies. We study how time restrictions (given by resource providers and users) influence the quality of schedules, and which scheme of grouping and ordering the tasks is the most effective for the batched scheduling of non-urgent workflows. Experiments with several types of synthetic and domain-specific sets of multiple workflows show that: (i) the use of information about time windows and deadlines leads to the significant increase of the quality of static schedules, (ii) the clustering-based scheduling scheme outperforms task-based and workflow-based schemes. This was confirmed by an evaluation of studied algorithms on a basis of the CLAVIRE workflow management platform.

Keywords: workflow scheduling, static scheduling algorithms, deadline-constrained workflows

1 Introduction

Scientific workflows, such as directed acyclic graphs (DAGs) of tasks, are widely used to perform multistage computationally intensive calculations. During execution of workflows in heterogeneous distributed systems, it is necessary to partition resources between users. This problem is solved by different workflow scheduling algorithms which are used in scientific workflow management systems (SWMS). Depending on a statement of a scheduling problem, these algorithms are applied to minimize makespan or cost of a given workflow, to ensure compliance with Quality of Service (QoS) requirements, to improve energy efficiency, etc.

To determine a particular scheduling algorithm for use in SWMS, one should take into account the following issues: (i) availability of information about tasks and workflows to be scheduled; (ii) properties of computational infrastructure of SWMS; (iii) characteristics of incoming rate of workflows; (iv) users' constraints for a resulting schedule; (v) computational complexity of an algorithm. For example, if the scheduling subsystem does not have estimations of execution times of tasks and predictions of workflows' incoming rate, it is reasonable to use simple dynamic heuristics (like Min-Min [1]). On the other hand, if we have the statistics of previous runs or parametric performance models [2] for all tasks, it is better to use static scheduling algorithms (like HEFT [3]) to take into account the structure of an entire workflow.

In this paper, we consider the scheduling of multiple workflows in a non-dedicated heterogeneous environment. A computational resource is called 'non-dedicated' (see, for example, [4]) when it is unavailable for calculations for certain periods of time (i.e., it has busy time windows). If the borders of time windows are known, they can be used by a scheduling procedure to improve the quality of a resulting plan. The following can be distinguished: (i) predefined time windows whose borders serve as input data to a scheduler, (ii) additional time windows which are reserved by a scheduler itself. Information about predefined time windows is available to a scheduler when some of the users sharing the resource pool are able to make an advance reservation [5][6] of resources. This technique is often used in non-dedicated environments to guarantee the provision of resources for a specific time interval [7]. Predefined time windows could also be determined by a scenario of infrastructure usage (e.g. computers belonging to a computer classroom will be busy during students' lessons) or by the terms of resource provision. The borders of additional time windows are set by the scheduler itself during the process of planning. Tasks should be assigned to resources taking into account previous assignments; hence the information about additional time windows could be used by a scheduling procedure to create a feasible schedule or to improve it (for example, in gap search algorithms [8][9] this information is exploited to increase the efficiency of scheduling of concurrent workflows).

We restrict ourselves to the problems for which properties (i)-(iv) are known before scheduling. Thus, the inputs of a scheduling algorithm include full information about workflows in the set, execution times of tasks using different resources, communication times between the tasks and (for each resource) borders of time windows available for calculations. Such scheduling problems arise during a periodical execution of prebuilt workflows on a fixed, non-extendable set of resources. As the same workflow is executed many times on the same set of resources, one can get sufficiently accurate estimates of execution and communication times. Combining such workflows into sets provides two main advantages: (i) more effective usage of computing power (because shapes of multiple DAGs are considered at once); (ii) an opportunity to tune a partition of resources between workflows in the set. The latter is useful when owners of workflows specify manifold requirements for the urgency of the results of calculations. In such a case, we should find a compromise between users' constraints in a resulting schedule. The example of such a problem is an application of scientific workflows in education [10] when students build their own workflows during practical work, and the result of execution should be received before a subsequent lesson.

For a given static set of resources, meeting the deadlines for all of the workflows in the set could be impossible. For this reason, we assume that users specify preferable finishing times (soft deadlines). The overview of related work in the area of multiple deadline-constrained workflow scheduling is given in Section 2. To estimate the quality of resulting schedule with account of different users' requirements, we introduce the unified metric incorporating levels of meeting the constraints as well as fairness of distribution of resources (Section 3). In Section 4, we explain the essence of the clustering-based approach, which generalizes common techniques of grouping and ordering the tasks of different workflows. In Section 5, we describe algorithms MDW-T, MDW-W, and MDW-C, implementing different clustering schemes (see our previous works [11], [12] for the detailed explanation). The experimental part of the study (Section 6) had two main objectives: (i) to quantify how the accounting of time windows and soft deadlines influences on the quality of schedules; (ii) to compare the quality of schedules provided by different algorithms for the sets of synthetic, benchmark and domain-specific workflows. To check the applicability of studied algorithms for SWMS, we also present the results of experiments with the CLAVIRE e-Science infrastructure platform.

2 Related work

The vast majority of works about deadline-constrained workflow scheduling is oriented to utility environments [13] (see, for example, [14], [15], [16], [17]). The defining characteristics of utility environments are a cost-based usage model and a logically unlimited resource pool (if there are insufficient resources to meet QoS constraints, a pool can be extended by starting additional virtual machines). Most algorithms for utility environments are intended for single workflow scheduling and do not take into consideration a competition between different users and different workflows for resources. There are fewer studies in the area of multiple workflow scheduling for utility environments. An example of the multiple workflow scheduling problem for utility environment can be found in [18], in which authors performed a comparative study of the effectiveness of three algorithms for prioritized budget- and deadline-constrained workflow ensembles in a cloud. Tolosana-Calasanz et al. [19] focus on the issues of data flow control of multiple pipelined workflow instances for on-demand resources under QoS constraints. The authors propose to maintain the desired data rate for data-intensive applications via the mechanism of token buckets.

In this work, we consider scheduling algorithms oriented to community environments [13]. These environments are characterized by a static structure of the set of resources and free-of-charge usage model. The differences between utility and community environments are reflected in different goals of scheduling process. In utility environments a scheduling algorithm should guarantee the satisfaction of QoS constraints for a dynamic resource pool; in community environments an algorithm deals with a static set of resources and it cannot ensure that all users' constraints will be met. Therefore algorithms oriented toward utility environments become inappropriate for community environments even if they consider multiple workflows with time constraints.

As in the previous case, the vast majority of works on scheduling in community environments are dedicated to a single workflow scheduling. A lot of efficient heuristics exist to minimize the makespan or maximize the efficiency of resource utilization during the execution of a single workflow (for example, the well-known HEFT [3] and its recent extension PEFT [20]). The algorithms intended for single workflow scheduling do not take into account the competition for resources between different workflows. Hence they cannot be used to find a compromise between time restrictions of multiple workflows in a set, which is the aim of our study.

Scheduling of multiple workflows in community environments is usually performed using one of three approaches. First of all, each workflow in a set can be represented as a single parallel task that executes on an arbitrary number of parallel processors. This method can be used for a homogeneous resource pool [21]. The second approach is based on a merging of all tasks from multiple workflows into one larger workflow, and on scheduling this workflow using any single workflow scheduling algorithm. This class of algorithms includes FPFT и FPCT [22], MWGS4 и MWGS2 [23]. Finally, the third approach uses a prioritization of workflows (or parts of workflows) in a set to find the most efficient order of scheduling. Bittencourt and Madeira, in [8], proposed an interleave algorithm where workflows are divided into groups of tasks (clusters) with the use of the Path Clustering Heuristic (PCH) algorithm, and then clusters from different DAGs are executed in a round robin fashion. All mentioned algorithms for scheduling multiple workflows in community environments do not consider a case when workflows in a set have soft deadlines, so tasks' assignments on resources are performed irrespective to the urgency levels of workflows. In contrast, our work is based on using information about time restrictions (both deadlines and time windows) to take into account users' preferences during scheduling.

The quality of a schedule for the case of multiple workflows is usually estimated by means of the following metrics: (i) the overall makespan [8] or its ratio to the optimal makespan [23] (it cannot be used to evaluate the level of meeting deadlines); (ii) the average makespan of a workflow [8][21][22] (it is appropriate for multiple instances of the same workflow rather than different workflows);

(iii) the fairness (or, otherwise, the unfairness [22]) of distribution of resources between workflows. The latter metric is used under the assumption that equal slowdowns [8][22] of any two different workflows have the same effect on the value of fairness. In the case of the presence of deadlines, this assumption cannot be applied because the value of fairness is influenced by the level to which the deadlines are met. In sum, the currently known metrics of schedule quality for multiple workflows are unsuitable to assess the level of meeting users' requirements, so it was necessary to develop a new metric – schedules' comparison criterion for multiple workflows with different deadlines. The description of this metric is given in Section 3.

3 An integral criterion of schedule quality for multiple workflows with soft deadlines

A mathematical formulation of a problem of multiple deadline-constrained workflows for a non-dedicated heterogeneous environment was introduced in our previous work (see the details in [11]). Given a set of N_{Wf} workflows $Wf = \{Wf_i\}$, $1 \le i \le N_{Wf}$ and N_R resources with different computational power. The workflow Wf_i is represented as the DAG of Nt_i tasks $Task_{ij}, 1 \le j \le Nt_i$. The dependencies between the tasks of i-th DAG are given by its adjacency matrix $E_i = \{e_{jk} \mid e_{jk} = 1 \text{ if } Task_{ij} \text{ is a parent of } Task_{ik}, e_{jk} = 0 \text{ otherwise}\}$, $1 \le j, k \le Nt_i$. The execution time of $Task_{ij}$ on the resource R_k, $1 \le k \le N_R$ is denoted as t_{ijk}. In this study, we supplement the model with times of data transfer between the tasks. Communication times are calculated by incorporating the following: (i) the matrix *Mcomm* of size $N_R \times N_R$, which contains the speeds of data transfer between resources; (ii) for each workflow Wf_i – the three-dimensional matrix $Data_{ijl}$ with the amounts of data transferred from $Task_{ij}$ to $Task_{il}$.

For i-th workflow, we define a tuple (ts_i, tc_i, d_i), where ts_i and tc_i – the start and the completion time of the Wf_i in the schedule, d_i – its soft deadline. The length of the planning period, T, is equal to a maximum deadline ($T = \max d_i$), so ts_i should be less than or equal to T. As resources are non-dedicated, for each resource R_k we specify the borders of Nw_k time windows unavailable for calculations $[ws_{kl}; wc_{kl}]$, $1 \le l \le Nw_k$, $0 \le ws_{kl}, wc_{kl} \le T$, $ws_{kl} \le wc_{kl}$. The goal of the scheduling process is to identify a mapping of the set of tasks to the set of resources with indication of the start time for each task ts_{ij} (finishing time of task is denoted as tc_{ij}). We assume that $ts_{ij} < T$ (a task cannot be started after the end of the planning period), so the following three types of schedules can be distinguished: (i) acceptable schedule – all tasks are finished before the end of the planning period: $\forall i, j \;\; tc_{ij} \le T$; (ii) partially acceptable schedule – all tasks are mapped to resources, but some of them are finished after the end of the planning period: $\forall i, j \;\; ts_{ij} < T$, $\exists j^* : tc_{ij^*} > T$; iii) incomplete schedule – some tasks are not mapped to resources.

As we schedule multiple workflows with different computational complexities and soft deadlines simultaneously, there is a need for a unified metric to compare resulting schedules. This metric (we call it the integral criterion and denote it as U) is aimed at estimating the two most important properties of a schedule: (i) the level to which the soft deadlines are met; (ii) the fairness of distribution of resources between workflows. We assume that $U \in [0;1]$ achieves its maximum value, 1, when all deadlines are met, and its minimum value, 0, when none of the tasks are assigned to a resource; so a better schedule will have a larger integral criterion.

As a set of resources is static, and cannot be expanded to meet users' requirements reliably, some workflows can violate their soft deadlines. The fine for violation of the deadline for the Wf_i is calculated as $Fine_i = \max\{tc_i - d_i, 0\}$. To calculate fines for an incomplete schedule, we assume that tasks without mapping to resources are executed consequently after the last of the scheduled tasks with average values of execution and communication times. The maximum fine for the *i*-th workflow, $Fine_i^{\max}$, is achieved when all of its tasks are not assigned to resources. We propose to use the relative average fine, $\overline{Fine}$, to estimate the level to which the soft deadlines are met:

$$\overline{Fine} = \frac{1}{N_{Wf}} \cdot \sum_{i=1}^{N_{Wf}} \frac{Fine_i}{Fine_i^{\max}} . \tag{1}$$

The relative average fine is intended to quantify the level of violating the deadlines for all of the workflows. This metric does not represent the variance of absolute values of fines for different workflows, i.e. it cannot measure the level of fairness of resource distribution. To estimate this level, we introduce the metric referred to as *Fairness* ($1 \le i, j \le N_{WF}$):

$$Fairness = 1 - \max_{i,j} \frac{\left| Fine_i - Fine_j \right|}{Fine_i^{\max}} . \tag{2}$$

Let X be a solution space (a set of possible schedules), $x_1, x_2 \in X$ – two schedules. Then, the integral criterion of quality of schedules for multiple workflows with soft deadlines $U : X \to [0;1]$ is defined as:

$$U = \rho_1 \cdot Fairness + \rho_2 \cdot \left(1 - \overline{Fine}\right), \tag{3}$$

where $\rho_1, \rho_2 \in [0;1]$, $\rho_1 + \rho_2 = 1$ – the weighting coefficients of corresponding criteria.

Then, a decision on preference of one schedule to another could be performed due to the following criterion:

$$x_1 \succ_X x_2 \Leftrightarrow U(Sched_1) > U(Sched_2) . \tag{4}$$

The expression (4) specifies the strict preference relation, because it has two necessary properties:

(i) irreflexivity – there are no $x_i \in X$, for which $x_i \succ x_i$;

(ii) transitivity – $x_i \succ x_j \wedge x_j \succ x_k \Rightarrow x_i \succ x_k$.

4 A clustering-based approach to static scheduling of multiple workflows with soft deadlines

4.1 General description of a clustering-based approach

The overview of the related work (Section 2) shows that there are three main approaches to schedule the sets of workflows: (i) task-based, when all tasks from all workflows are merged into a large workflow; (ii) workflow-based, when the goal of a scheduling process is to find the order of scheduling of independent workflows; (iii) clustering-based, when each workflow is divided into groups of tasks that are further scheduled according to a certain order. For all of these, the main stages of a planning process are the same: (i) clustering – a creation of groups (clusters) of tasks, (ii) ordering – a search for consistent order of clusters, and (iii) scheduling – a search for final mapping of tasks on resources for all clusters according to the order. In a task-based approach, a single task represents a single cluster, and an ordering coincides with scheduling. In a workflow-based approach, clusters are created from independent workflows, and the goal of an ordering is to identify an order of workflows. Therefore, we can conclude that the task-based and the workflow-based approaches are special cases of the clustering-based approach.

Formally, the general problem of clustering of the set of workflows can be described as follows: to find a division of a set of tasks $\{Task_{ij}\}$ on subsets of N_C clusters C_k, $1 \le k \le N_C$, so that any task belongs to one and only one cluster. If tasks in the same cluster cannot be executed in parallel (like in the PCH algorithm), there is only one order of scheduling (and execution) of tasks within the cluster. In general, the set of clusters should satisfy the constraint of correctness to be used in a scheduling algorithm.

The result of clustering is correct if there is an uncontroversial order of clusters such that for each task all of its predecessors belong to its cluster or to the one of the previous clusters. More formally, let $Pred(A)$ be a set of predecessors for the task A in a DAG. The set of clusters will be correct when the following ordered set of indexes of clusters is found: $Ord = \{k_1, k_2, \dots, k_i, \dots, k_{N_C}\}$, $k_i \ne k_j$ for $i \ne j$, $1 \le k_i \le N_C$ such that $\forall i \in \{1, \dots, N_C\}$, $\forall A \in C_{k_i}$, $\forall B \in Pred(A)$ $B \in C_{k_1} \cup \dots \cup C_{k_{i-1}} \cup C_{k_i}$. If the set is correct, one can schedule cluster by cluster due to the order of their indexes in *Ord* without violating the precedence constraints. In summary, a scheduling algorithm for multiple workflows is uniquely determined by: (i) a method of clustering; (ii) a method of ordering the clusters; (iii) a method of ordering the tasks inside clusters.

4.2 Algorithms for scheduling multiple workflows with soft deadlines

One of the goals of this study was to compare the effectiveness of different variants of clustering schemes for scheduling multiple workflows with soft deadlines. Algorithms implementing task-based and workflow-based schemes were proposed in our previous work [11], and here we give a brief description.

I. Task-based scheme (MDW-T algorithm).

The clustering phase for the task-based scheme is trivial (each task represents a single cluster), so it is enough to specify how to order the tasks and how to schedule them. The main idea of the MDW-T (**M**ultiple **D**eadline-constrained **W**orkflows – **T**ask-based) algorithm is a prioritization of tasks according to their subdeadlines. Similar prioritization is often used in single deadline-constrained workflow scheduling (see, for example, [24], [25], [26]). Subdeadlines in MDW-T are calculated proportional to an accumulated computational amount (or average execution time) of a particular task (we call this parameter a weight). When accumulated weights are calculated, the sub-deadline of the task with the maximum weight is set to a deadline of the whole workflow, and other sub-deadlines are found in proportion. The scheduling queue of tasks is formed, with sub-deadlines in ascending order.

Let U be the set of tasks which weights are already calculated, $-L = \{l_i\}$, $1 \le i \le |L|$ – the sequence of children tasks for tasks in U which weights are not calculated yet, P – the set of parents of $Task_{ij}$, w_{ij} – the weight of $Task_{ij}$, $\overline{t_{ij}}$ – the average execution time of $Task_{ij}$, $\overline{c_{ijk}}$ – the average communication time between $Task_{ij}$ and $Task_{ik}$, d_{ij} – the subdeadline of $Task_{ij}$. Then prioritization of the tasks in MDW-T can be described as follows (Table 1).

```
1: for $i = 1..N_{WF}$                                        // for all workflows
2:     $U \leftarrow \{Task_{ij} : \forall k\ 1 \le k \le Nt_i\ e_{kj} = 0\}$      // $U$ include tasks without parents in $Wf_i$
3:     $L \leftarrow \{Task_{ik} : Task_{ij} \in U, e_{jk} = 1\}$       // $L$ include tasks which parents are in $U$
```

```
4:   for each $Task_{ij} \in U$ $w_{ij} = \overline{t_{ij}}$            // calculate weights of initial tasks
5:   While $L \neq \varnothing$ repeat steps 6-13
6:     $j^* = j : l_1 = Task_{ij}$                                      // j* is the index of first task in L
7:     $L = L \setminus \{Task_{ij^*}\}$                                  // the current task is removed from L
8:     $P \leftarrow \{Task_{ij} : e_{jj^*} = 1\}$                        // P include parents of the current task
9:     If $P \subseteq U$ then                                          // if weights of all parents of $Task_{ij^*}$ are known,
10:      $w_{ij^*} = \max_{j \in P}\{w_{ij} + \overline{c_{ijj^*}}\} + \overline{t_{ij^*}}$   // we can calculate its weight
11:      $U \leftarrow U \cup \{Task_{ij^*}\}$
12:      $\forall Task_{ij} \notin L : e_{j^*j} = 1$ $L \leftarrow L \cup \{Task_{ij}\}$   // add children of current task to the end of L
13:    Else $L \leftarrow L \cup \{Task_{ij^*}\}$                        // else move the task to the end of the sequence
14:  $j^* = j : \forall j,k$ $w_{ij} \geq w_{ik}$                          // j* is the index of task with maximum weight
15:  $d_{ij^*} = d_i$                                                   // its subdeadline is set to the deadline of $Wf_i$
16:  $\forall Task_{ij} \in Wf_i, j \neq j^*$ $d_{ij} = \frac{w_{ij}}{w_{ij^*}} \cdot d_{ij}$   // other subdeadlines are found in proportion
17:  Create a priority queue of all tasks from all workflows (tasks are placed in the queue in
     ascending order of their subdeadlines)
```

Table 1. Prioritization of the tasks in the MDW-T algorithm

where $P_{ij} = \{\ Task_{ip}\ \}$, $p \in 1,\ldots,Nt_i$ – the set of parent tasks for $Task_{ij}$, k_{ip} - the index of resource of $Task_{ip}$ in the schedule. If $P_{ij} = \varnothing$ then earliest start time of $Task_{ij}$ is set to zero.

The possible start time of $Task_{ij}$ on the resource R_k depends on EST_{ijk}, time of execution of the task t_{ijk}, and borders of busy time windows:

$$PST_{ijk} = \begin{cases} EST_{ijk}, \text{if } \forall l \in 1,\ldots,Nw_k \left(EST_{ijk}; EST_{ijk} + t_{ijk}\right) \cap \left[ws_{kl}; wc_{kl}\right] = \varnothing \\ \min\limits_{l} wc_{kl} : wc_{kl} > EST_{ijk}, wc_{kl} + t_{ijk} \leq ws_{k,l+1} \text{ for } l < Nw_k, \text{otherwise}. \end{cases} \quad (6)$$

If $Task_{ij}$ can be executed at the earliest start time, the possible start time is equal to EST_{ijk}. Otherwise, PST_{ijk} is equal to the earliest possible time for the start of the task. When PST_{ijk} is equal to T (the end of the planning period), $Task_{ij}$ cannot be assigned to resource R_k. The final destination for $Task_{ij}$ is the resource R_{k*} where k^*: $PST_{ijk^*} + t_{ijk^*} \leq PST_{ijk} + t_{ijk}$ $\forall k \in 1,\ldots,N_R$ and $PST_{ijk^*} < T$. If such value of k^* cannot be found, $Task_{ij}$ will not be assigned to any resource.

The computational complexity of MDW-T is identical to that of the HEFT algorithm and then is equal to $O\left(Nt^2 \cdot N_R\right)$ where $Nt = \sum_i Nt_i$.

II. Workflow-based scheme (MDW-W algorithm).

For the workflow-based scheme, a single workflow represents a single cluster, and the goal is the prioritization of workflows. The corresponding algorithm is denoted as MDW-W (**M**ultiple **D**eadline-constrained **W**orkflows – **W**orkflow-based). In [11], we proposed the staged scheme of ordering the workflows, when it is performed iteratively with the use of criterion of prioritization – a certain metric applicable to workflow schedule. The extreme value of the criterion is used for a choice of subsequent workflow that will have access to resources. In this work, we use the minimum of reserved time $Rt_i = \max\{d_i - tc_i, 0\}$ as the prioritization criterion. The process of ordering according to the staged scheme consists of N_{Wf} stages. At stage 1, we search for the schedules of all workflows independently (by providing all resources to each of them in turn) and choose for a scheduling the workflow with an extreme criterion value. Next, this process is repeated for remaining workflows.

Let *U* be a set of scheduled workflows, *L* – a set of workflows to be scheduled, *I* – a set of current busy time windows, $S_i(I)$ – the schedule for Wf_i and time windows *I*, $PC(S_i)$ – the prioritization criterion for schedule $S_i(I)$. The description of MDW-W algorithm is presented in Table 2.

```
1:    $U = \varnothing$, $L = \varnothing$
2:    Add predefined time windows to $I$
3:    For $i = 1..N_{Wf}$  $L \leftarrow L \cup Wf_i$
4:    For all $Wf_i \in L$ get the schedule $S_i(I)$.
5:    Find $i^* = \{i : \forall Wf_i, Wf_j \in L, i \neq j \;\; PC(S_i) \leq (\geq) PC(S_j)\}$
6:    $U \leftarrow U \cup \{Wf_{i*}\}$, $L \leftarrow L \setminus \{Wf_{i*}\}$.
7:    Fix busy time windows for $Wf_{i*}$ in $I$ in accordance to $S_{i*}(I)$
8:    Repeat steps 4-7 while $L \neq \varnothing$.
```

Table 2. The description of MDW-W algorithm

Scheduling of a single workflow in MDW-W can be performed with the use of any single workflow scheduling algorithm; we use MDW-T for this purpose, as it takes into account the sub-deadlines of tasks during their ordering. In such a case the computational complexity of MDW-W is $O\left(N_{Wf}{}^2 \cdot Nt_{\max}^2 \cdot N_R\right)$ where $Nt_{\max} = \max_i Nt_i$.

III. Clustering-based scheme (MDW-C algorithm).

A variety of possible methods of clustering the tasks of several workflows can be divided into two groups: (i) joint clustering, when a single cluster can contain tasks from different workflows; (ii) independent clustering, otherwise. In this study, we focus on the latter method because it provides obvious way of parallelizing the clustering process (time of clustering can be significant, especially for the sets of big sizes). As mentioned before, we should specify how to perform three phases of scheduling process (grouping, ordering and mapping) for clustering-based algorithm, which we denote as MDW-C (**M**ultiple **D**eadline-constrained **W**orkflows – **C**lustering-based). The brief description of this algorithm can be found in [12]. In this paper, we give a more detailed explanation of the clustering process. MDW-C combines the ideas of task-based and workflow-based schemes, because it uses both prioritization of the tasks according to their subdeadlines, and prioritization of the clusters as parts of workflows. The main phases of MDW-C can be described as follows.

1. Clustering.

MDW-C uses independent clustering, so a particular cluster can contain tasks belonging to only one workflow, and the clustering process can be performed in parallel for all the workflows in the set. A cluster, C, is characterized with: (i) the deadline $d(C) = \max d_{ij}$, (ii) the start time $ts(C) = \min ts_{ij}$, (iii) the weight $w(C) = \sum \overline{t_{ij}}$, and (iv) the length $l(C) = d(C) - ts(C)$, where $(i, j): Task_{ij} \in C$, $\overline{t_{ij}}$ – an average execution time of $Task_{ij}$. In MDW-C, we want to rotate parts of different workflows during scheduling. To determine these parts, we try to take into account both computational complexities of tasks and the shape of corresponding DAG (using the values of weights and lengths, correspondingly). The clustering has two complementary goals: (i) to create clusters with approximately equal weights, and (ii) to combine the same cluster tasks with similar subdeadlines to be executed in parallel. The advantage of the latter is that tasks with similar subdeadlines will have access to resources at the same time, and then they will have a chance to finish their execution at a similar time. Otherwise, for example, if tasks with similar subdeadlines will be placed in two different clusters, one part of them will be mapped to resources later than the other, resulting in the delay of the whole workflow. From the other side, equalization of clusters' weights is a way to prevent a creation of clusters with small (like in MDW-T) or heavy (like in MDW-W) weights. The clustering process is illustrated in Figure 1.

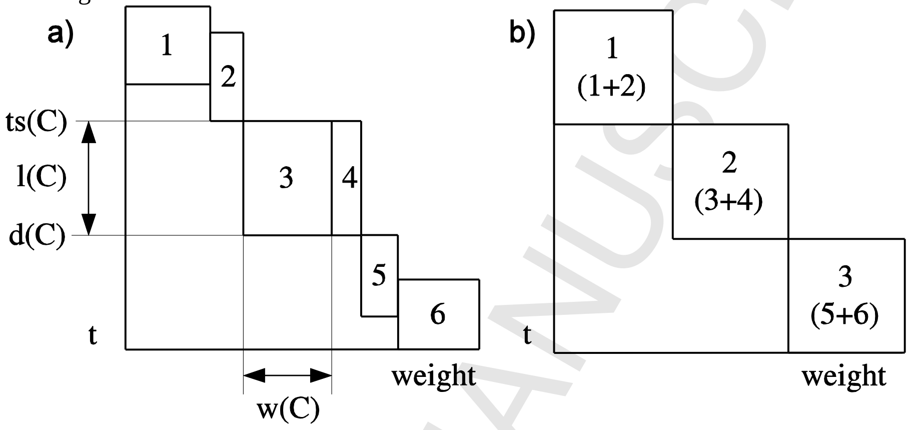

Figure 1: Overview of the clustering process

Each cluster in Figure 1 is depicted as a rectangle. The width of a rectangle represents the weight of a cluster, and the height represents the length of a cluster. Clusters are shown from the left to the right in ascending order of their start times. As we can see in Figure 1a, the primary variant of clustering has the following disadvantages: (i) the clusters have vastly different weights; (ii) the tasks with similar subdeadlines are in different clusters. The clustering process aims to overcome these drawbacks, and the ideal result is depicted in Figure 1b) –the weights of clusters are identical and the sum of lengths of clusters is minimal.

The clustering phase of MDW-C is based on a search for balance between the mean difference of weights of clusters $\overline{\Delta w}$ and the sum of lengths of clusters L. A decrease of the mean difference of weights helps to distribute weights more uniformly, and a decrease of the total length of clusters leads to the placement of tasks with near sub-deadlines in the same clusters. To form clusters with approximately equal weights and lengths, we try to minimize $f(SetC) = \overline{\Delta w} / \overline{\Delta w}_{\max} + L / L_{\max}$, where $SetC = \{C_1, \ldots, C_k, \ldots, C_{N_C}\}$ is a current set of clusters, and N_C is a number of clusters. We define maximum possible values of $\overline{\Delta w}$ and L as $\overline{\Delta w}_{\max} = \sum_k w(C_k)/2$, $L_{\max} = \sum_{i,j} \overline{t_{ij}}$ for $Task_{ij} \in SetC$.

Figure 2 represents an example of the iterative process of clustering (the number under the rectangle denotes weight, the number within the rectangle denotes length). Initially (Figure 2a), each task is in a separate cluster. The indexes of clusters are arranged in ascending order of their sub-deadlines. Such arrangement guarantees the correctness of the initial set of clusters. In addition, merging two neighboring clusters provides the correct set of clusters. During each iteration (Figure 2b-d), we make an attempt to merge the current cluster with the previous or the next cluster. An index of current cluster varies from 2 to $N_C - 1$. For each attempt of merging, we calculate a new value of *f*.

We perform the merger if *f* decreases (clusters to be merged are determined by the minimum value of *f*). An index of current cluster increases if it was not involved in a merger. For example, during iteration 1 (Figure 2a-b) the current cluster is C_2 (it contains task B), and *f* decreases more significantly when we merge A and B. After the merger the cluster C_1 contains tasks A and B, C_2 contains task C, the index of the current cluster does not change (it is equal to 2). The results of the remaining iterations are shown in Figure 2c-d. The clustering continues in a cyclic manner, while there is at least one merger when passing from C_2 to C_{N_C-1}. Hence we gradually change the current cluster from C_2 to C_{N_C-1} trying to perform a merger, and clustering stops if no mergers are performed after that. Otherwise (if at least one merger was performed) we repeat the process from the beginning.

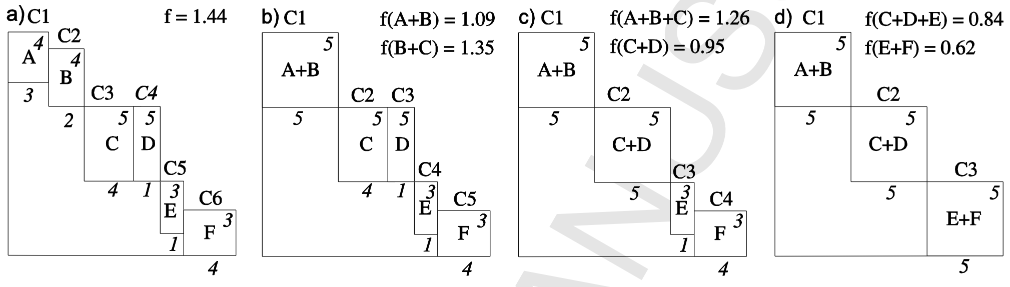

Figure 2. Clustering phase (an example)

If we perform a clustering for the workflow with N tasks, the maximum number of mergers will be $N-1$. For one merger we should calculate $f(SetC)$ two times, and one calculation takes $N_C^2 + N_C$ operations where N_C is a current number of clusters. As for a maximum number of mergers, the number of clusters consequently decreases from N to 1, the total number of operations is $\sum_{i=0}^{N-1}\left((N-i)^2+(N-i)\right) \sim N^3$. Since we perform a clustering for all workflows independently, the computational complexity of this procedure is $O\left(N_{Wf} \cdot Nt_{\max}^3\right)$.

2. Ordering and scheduling.

When all workflows are divided into clusters, one should find the order of clusters and schedule them according to this order. Firstly, we determine a workflow with minimum value of reserved time (like in MDW-W algorithm). Secondly, we choose a cluster from the candidate set *CS* of this workflow. *CS* contains unscheduled clusters whose parents are already scheduled. The choice of the cluster is performed according to the maximum of the following metric:

$$M(C) = \rho_{wl} \cdot \frac{w(C)/l(C)}{wl_{\max}} + \rho_d \cdot \frac{d_{\min}}{d(C)}. \qquad (7)$$

This metric has two components: (i) the normalized ratio of the cluster's weight to its length, (ii) the normalized deadline where ρ_{wl} , ρ_d are weighting coefficients, $wl_{\max} = \max w(C)/l(C)$, $d_{\min} = \min d(C)$, where $C \in CS$.

Due to the essence of the clustering method, tasks in a cluster are sorted in ascending order of their sub-deadlines. Scheduling of tasks of the cluster is performed according to this order with account of time windows. Therefore, the MDW-T algorithm is applied to schedule a particular cluster. After scheduling, we update the values of start times, deadlines and lengths for all successors of the current cluster and the candidate set *CS*. The process repeats until all clusters are scheduled.

5 A comparative study of the effectiveness of algorithms for scheduling multiple workflows with soft deadlines

The experimental part of the study aimed to achieve the following: (i) to measure the benefits of accounting time windows and soft deadlines in a scheduling algorithm; (ii) to compare the quality of schedules provided by algorithms implementing various clustering schemes. We estimate the quality of resulting schedules using integral criterion *U* (described in Section 3). We use equal weighting criteria $\rho_1 = \rho_2 = 0.5$ to calculate *U* as well as $\rho_{wl} = \rho_d = 0.5$ in MDW-C algorithm. The algorithm MDW-W uses MDW-T as an algorithm for single-workflow scheduling.

For a given set of input data (information about resources, network bandwidth and workflows in a set), a scheduling algorithm provides an execution plan, which we will call “basic schedule”. To create a basic schedule, an algorithm assumes that execution times of different tasks and communication times between different resources are exact and cannot vary over time. In the subsections 5.1, 5.2 and 5.3, we compare the quality of basic schedules without real runs of tasks in a distributed system. These results are important to analyze the behavior of algorithms at the different levels of competition for resources between workflows in a set. When basic schedules are used in a distributed system, they cannot be executed precisely due to time-varying performances of resources and possible task failures. The values of start and completion times of tasks, which are taken during their execution, form a schedule that we call “actual schedule”. In subsection 5.4, we describe our experiments that aimed to achieve the following: (i) to estimate the difference between basic and actual schedules, and (ii) to make the conclusions about applicability and comparative effectiveness of MDW-T, MDW-W and MDW-C in a distributed environment.

5.1 Synthetic test sets

This set of experiments aimed to observe the values of the integral criterion (and its components, fine and fairness) for algorithms with and without support of time windows. We compare the values obtained by two task-based algorithms: MDW-T and Min-Min. In fact, when all workflows in the set have identical deadlines, MDW-T forms a schedule on the same principles as Min-Min: it chooses for a scheduling the task with the soonest sub-deadline, one that will have the minimum completion time. The difference between them is that MDW-T takes into account time windows during task assignment to resources and Min-Min does not. We assume that the tasks in predefined windows have the highest priority, so they can interrupt the execution of lower priority tasks (assigned by the algorithms). Min-Min (as opposed to MDW-T) does not have information about borders of busy time windows, and it can map a task to a resource that does not provide enough time before the beginning of the next window. In such a case, we reschedule this task when it is interrupted. The goal of the experiment is to measure how the accounting of time windows by a scheduling algorithm influences the quality of the schedule.

To obtain the test data for this experiment, we used the DAGGEN [27] generator of synthetic workflows and the self-developed generator of resources. The description and the values of parameters used for generation are presented in Table 3. The size of a set of workflows N_{WF} varies from 20 to 100 in increments of 20. A particular set contains randomly chosen workflows with 5, 10, 20 or 50 tasks per workflow. The shape of a particular DAG is determined by the values of its width, density, regularity and jump (the detailed description of these parameters can be found in the documentation of DAGGEN). Obtained files with DAG descriptions are supplemented with the values of computational costs of tasks. These costs are chosen so that the execution time of a task varies in a range 300-43200 seconds for a given range of resources’ performances. For the experiments we use resources with five different performances (performances differ by no more than three times). The basic length of planning period T_0 is equal to 24 hours. The number of resources for each experiment is chosen so that the current set of workflows can be executed on a current set of resources in a time T_0 with the level of utilization of resources more than 80%. It means that different workflow sets have approximately the same conditions of access to resources.

Parameter name	**Parameter values**
Parameters of workflows	
Set size	20; 40; 60; 80; 100
Task computational cost (GFLOP)	15000-216000
Number of tasks per workflow	5; 10; 20; 50
Width of the DAG	0.1; 0.2; 0.8
DAG density	0.2; 0.8
DAG regularity	0.2; 0.8
DAG jump	1, 2, 4
Parameters of resources	
Resource performance (GFLOPS)	5-50
Part of non-dedicated time per resource (k)	0.25; 0.5; 0.75
Number of time windows per resource	1-10

Table 3. Parameters of generation of workflows and resources

To study the influence of using the information about predefined time windows on the quality of a schedule, we try to simulate the advance reservation of resources for different levels of their availability. We assume that predefined time windows are occupied with tasks similar to those included in synthetic DAGs. As for this set of experiments we use workflows of different sizes with widely varying computational cost of tasks, we vary a count of predefined time windows and their lengths for a given part of occupied time per resource. We performed the experiments for the total length of busy time windows per resource equal to 25%, 50%, 75% from T_0. To keep the total time available for scheduling equal to T_0, we expand the length of the planning period to $T = T_0 + k \cdot T_0$ where $k = 0.25; 0.5; 0.75$ – the portion of time occupied by predefined windows. Also, we vary the number of time windows per resource *Nw* from 1 to 10. The borders of time windows are assigned randomly with the range $[0;T]$. The deadlines for all workflows are set to the value of *T*. We performed five runs of scheduling for each combination of *k*, *Nw* and N_{WF}. Then, we performed averaging: (i) by runs, (ii) by sizes of sets of workflow, and (iii) by *k*. The results are presented in Figure 3.

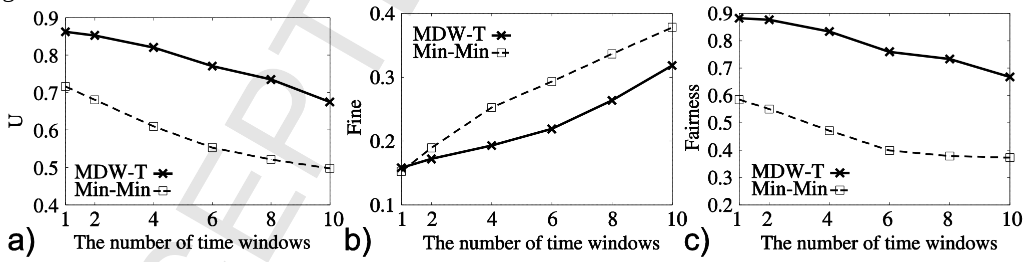

Figure 3. Integral criterion (a), fine (b) and fairness (c) for synthetic test sets

MDW-T takes advantage of the information about time windows, and it exceeds Min-Min by an average of 18% (for *k*=0.25 the average advantage is equal to 21%, for *k*=0.5 – 22%, and for *k*=0.75 – 14%). The value of integral criterion (Figure 3a) decreases with the increase of the number of time windows for both algorithms. The reason is that the planning period for each resource is fractionized on approximately $2 \cdot Nw$ busy and free intervals of different lengths, and the average size of a free interval decreases with the increase of *Nw*. The disadvantage of Min-Min is primarily due to the low fairness of its schedules (Figure 3b,c). It means that Min-Min provides non-uniform distribution of fines in a non-dedicated environment. It cannot predict which tasks will be interrupted after their assignment to resources, and workflows with such tasks experience more significant penalties than others.

5.2 Cybershake and Genome test sets

For the second set of experiments, we use two types of benchmark workflows (Cybershake and Genome) available on the Pegasus website [28]. These sets are often used to compare the quality of schedules obtained by different algorithms (see, for example, [24][29]). Information about the task dependencies, the estimates of execution times and the amount of transferred data are represented in DAX format. Since DAX files include a single time estimate for each task, we consider this value as a time of execution of this task in the slowest resource of a particular set, and calculate other estimates in proportion to the performances of resource types. A start time and a deadline for each workflow are assigned randomly to simulate the case of different time restrictions given by users. To measure how the accounting of soft deadlines influences the quality of schedules, we compare MDW-T, MDW-W and MDW-C using the PCH algorithm [8]. For the latter algorithm, we perform the mapping of tasks using the information about available time windows to neutralize the disadvantage described in subsection 5.1. The portion of time unavailable for calculations, *k*, is equal to 0.25 for all experiments; *Nw* is chosen randomly from the range 1 to 10.

Each test set (for example, Genome) includes workflows of 50, 100, 200-1000 tasks. One run of scheduling is performed for a set of 20 workflows of the same size. As we assign the start time and the deadline for each workflow in the set randomly, we should guarantee that the value of $l_i = d_i - ts_i$, $1 \le i \le N_{Wf}$ is sufficient for execution of a single workflow without violating the deadline. For this purpose, we use the preliminary runs to obtain the value of sL – the time enough to execute a single workflow of current size using a current set of resources. Then, we assign d_i and ts_i so that $sL \le l_i$.

The goal of the experiments was to study the behavior of algorithms for the different levels of competition for resources. We simulate these levels, varying the value of l_i on the same set of resources. We assume that $l = l_i$ is the same for all workflows in the set and gradually increase l from sL to $l_{\max}$ during $n_{\max}$ steps. The value of $n_{\max} = 12$ was taken during our experiments as sufficient to achieve the value of *U* = 1 at least for one of the algorithms. To calculate l for each step, we use the expression $l = sL + (n-1) \cdot c \cdot sL$ where n – the index of step (from 1 to $n_{\max}$), *c* – the coefficient determining the rate of growth of l per step (we used value of 0.125). After calculating the value of *l*, we randomly assign the start time for each workflow in the range [0;*l*], set their deadlines to $ts_i + l$, and set *T* to the maximum of the deadline. In such conditions, when l is equal to *sL* (step 1), the value of integral criterion is usually no more than 0.3 due to a peck of unscheduled tasks, while $l = l_{\max}$ (step $n_{\max}$) allows execution of all workflows in the set without violating the deadlines.

We perform five runs of a scheduling procedure for each step. Thus, the number of experiments for one set of workflows (e.g., 20 Genome workflows each with 200 tasks) is equal to 60, and the total number of experiments for one test set (e.g., Genome) is equal to 660. The presented results are taken by averaging the values for the sets of workflows of different sizes. Also, we modify the size of the resource pool by increasing the size of the workflow in the set (for example, a set of workflows with 50 tasks is scheduled using four types of 16 resources, and a set of workflows with 1000 tasks – using eight types of 128 resources).

Figures 4 shows the resulting values of the integral criterion for Cybershake and Genome test sets.

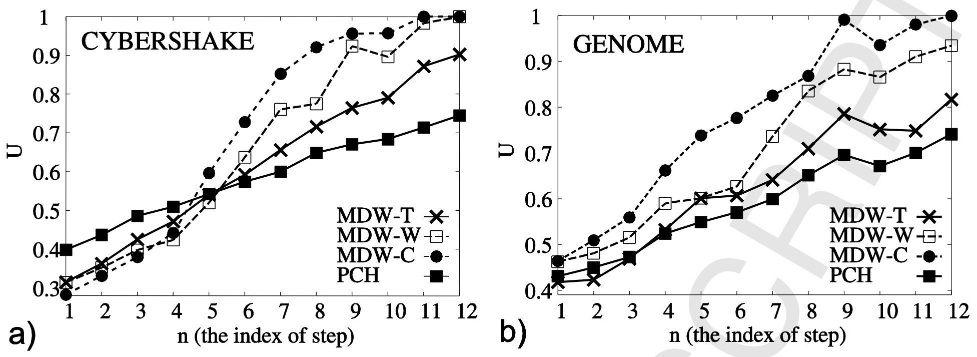

Figure 4. Integral criterion for Cybershake (a) and Genome (b) test sets

The advantage of MDW-C in comparison with MDW-W, MDW-T, PCH is as follows: for Cybershake test set – 7%, 17%, 11%, for Genome test set – 7%, 14%, 19%, respectively. For early steps, the value of *l* is insufficient for timely execution of all workflows, so we observe a large number of unscheduled tasks, high values of fines and low values of integral criterion for all three algorithms. PCH demonstrates the lowest growth rate of *U*, and the worst absolute values of *U*, *Fine* and *Fairness*, for later steps when *l* becomes sufficient for non-violation of most deadlines. The comparative analysis of these metrics enables us to conclude that using the values of soft deadlines during prioritization of the tasks of multiple workflows helps to obtain more qualitative schedules. In particular, when MDW-C achieves the maximum of *U*, PCH has a value of less than 0.75 with an average fine of more than 0.12 for all experiments. Thus, under the same conditions, using PCH leads to a delay of approximately 12% of all computing load, while MDW-C schedules all workflows without violating the deadlines.

Figure 5 represents the execution time of MDW-T, MDW-W and MDW-C for different sizes of GENOME workflows (from 50 to 1000 tasks). The results confirmed the theoretical estimates of the computational complexity described in Section 4.2.

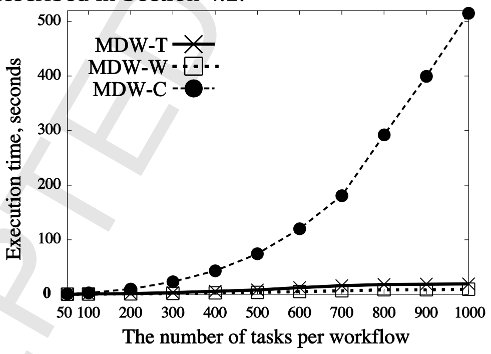

Figure 5. The execution time of MDW-T, MDW-W and MDW-C for GENOME workflow

5.3 Domain-specific test sets: flood forecasting in Saint Petersburg

As a practical example of the application of multiple deadline-constrained workflow scheduling algorithms, we consider the problem of flood forecasting in St. Petersburg [30]. To tune the parameters of a forecast, one should run the corresponding workflow several times with different inputs, which are obtained from the previously calculated forecasts. These runs could be executed in parallel, so we can combine workflows into sets to reduce overall execution time of the tuning procedure and to effectively fill available resources with blocks from different workflow instances. In this paper, we use two different workflows in such a scenario: (i) a workflow for the ensemble flood forecasting in Saint Petersburg (WF1); (ii) a workflow to calculate a field of sea waves on the Baltic Sea for use in a flood prevention system (WF2). In order to determine the robustness of parameters' estimates, some of the tasks of these workflows can be sampled, i.e. they can be started *M* times with different values of noise in input data using the Monte Carlo method. Hence if one has a workflow with *N* tasks, and *K* of them could be sampled, the total number of tasks in an abstract workflow after the end of sampling procedure will be equal to $N + K \cdot (M - 1)$.

We compare the values of the integral criterion obtained with MDW-T, MDW-W and MDW-C algorithms for the sets of modeling workflows (size of set is equal to 20) for different values of sampling parameter *M*. For WF1, *M* varies from 1 to 100 with step 10 (the size of a workflow varies from 8 to 503 tasks), and for WF2 *M* varies from 1 to 128 with step 16 (the size of a workflow varies from 24 to 264 tasks). The set of resources includes resources of three types; the performances of resources of these types related to each other are 1/0.8/1.5. As we used workflows of different sizes, we consequently increase a number of resources for each value of *M*, keeping the same ratio between the numbers of resources of different types.

For WF1, experiments were performed using a technique described in Section 5.2, so for a particular *M* the scheduling procedure was run five times for each of the 12 steps. For each set of workflows with type WF2, we experimentally determined the borders [*a*;*b*], where *a* – the minimum value of *T*, which provides $U \geq 0.5$ for all three algorithms, *b* – the minimum value of *T*, which provides $U = 1$ for at least one algorithm. In this way, we indicated the range of greatest interest to analyze the behavior of algorithms. We performed six steps, with *T* varying from *a* to *b* (five runs per step). Figure 6 demonstrates the resulting average values of integral criterion for WF1 and WF2. The average advantage of MDW-C is 6% and 11% for WF1, 9% and 17% for WF2 (in comparison with MDW-T and MDW-W, respectively). The superiority of MDW-T over MDW-W for these test sets could be explained by the fact that the sampling procedure creates the bags of independent tasks inside a workflow. Under such conditions, the dynamic rotation of tasks from different workflows in a priority queue (like MDW-T does) is more effective than scheduling workflows one after another (like in MDW-W).

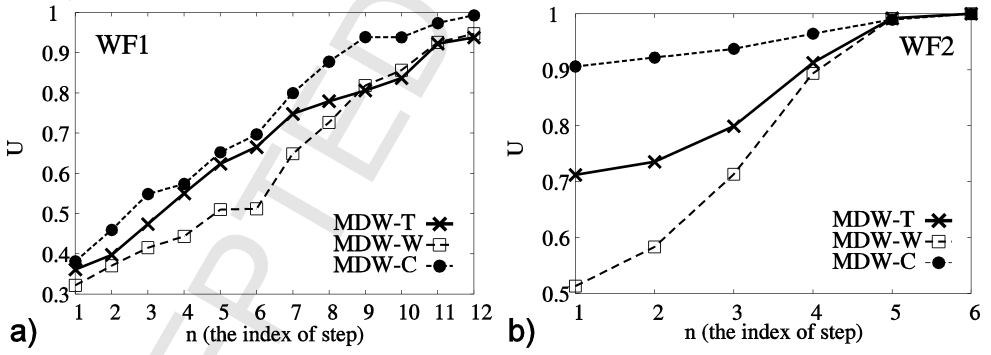

Figure 6: Integral criterion for WF1 (a) and WF2 (b)

5.4 An experimental study of the effectiveness of algorithms on the basis of scientific workflow management system

To estimate the applicability of studied algorithms for SWMS, we use the CLAVIRE [31] e-Science infrastructure platform. CLAVIRE is a tool for data-driven computing to run workflows using different types of computational environments (including Grids and clouds). Like other SWMS, CLAVIRE provides the opportunities of automatic mapping of tasks to available resources with the use of its local scheduling procedure. This procedure is based on task-by-task scheduling and does not support static schedules for several workflows at once. To overcome this issue, we developed a service *SchedulerService* as an intermediate between *ExecutionBroker* service (which is responsible for starting of tasks) and *WFSched* application (which provides static schedules using MDW-T, MDW-W or MDW-C algorithm). During its first call, *SchedulerService* gets a schedule for a set of workflows passed from *ExecutionBroker*, and forms the queues of tasks for all resources of a computational system. The subsequent calls of the scheduler do not create new schedules, but only update the queues due to the previous plan. Support for time windows [11] (periods of time when resources are unavailable for execution of workflows) is provided by both *SchedulerService* (when it works with queues) and *WFSched* (when it creates a schedule).

We selected the Montage workflow for experiments with CLAVIRE, as it is a de-facto benchmark to compare workflow scheduling algorithms. We embedded nine basic Montage tasks into a base of packages of CLAVIRE, and made a description of a whole workflow using the EasyFlow script language (some details of the embedding procedure can be found in [31]). A size of the test workflow was equal to 20. As the infrastructure for experimental runs, we deployed 20 VMs (Oracle VirtualBox, Windows 7) on two blade servers (32xIntel Xeon E5-2650 2.00 GHz, 96 Gb RAM). Heterogeneity of resources was emulated with a software setting of VirtualBox, which can be used to vary processor performance. Also, we developed a *CommCreator* application to aggregate the information about task start and completion times on the nodes. Before the beginning of the experiments, we performed 20 preliminary runs for each task of Montage to compute average estimations of execution times to use them in a scheduling procedure. We run the sets of identical workflows on different input data, with each set ranging from 40 to 120 with a step of 20. The deadline was set the same for all of the workflows. A particular value of the deadline was determined using the technique described in Section 5.2 as the first value of l for which all algorithms provide basic schedules with U = 1. As one goal of the experiments was to compare basic and actual schedules, we observe not only values of integral criterion, but several other metrics: (i) part of delayed tasks; ii) average relative delay of a workflow; iii) ratio of actual completion time of a set t_{real} to a basic completion time t_{sch}, iv) ratio of t_{real} to deadline, and (v) average reserved time R (in proportion to the length of the planning period):

$$R = \overline{Rt}/T = \sum_{i=1}^{N_{Wf}} \frac{Rt_i}{N_{Wf}} \cdot \frac{1}{T} . \qquad (8)$$

Set size	Integral criterion			Average reserved time (part of *T*)			Part of delayed tasks		
	MDW-T	MDW-W	MDW-C	MDW-T	MDW-W	MDW-C	MDW-T	MDW-W	MDW-C
40	1	1	1	0.295	0.415	0.303	0.725	0.518	0.675
60	1	1	1	0.073	0.086	0.09	0.71	0.7	0.7
80	0.93	0.98	1	0	0.004	0.083	0.73	0.65	0.375
100	1	1	1	0.083	0.061	0.07	0.70	0.88	0.74
120	0.99	0.99	1	0.02	0.023	0.083	0.91	0.775	0.83

Set size	Average delay time of a workflow (part of *T*)			t_{real} / t_{sch}			t_{real} / *deadline*		
	MDW-T	MDW-W	MDW-C	MDW-T	MDW-W	MDW-C	MDW-T	MDW-W	MDW-C
40	0.0297	0	0.001	1.085	0.99	1.00	0.757	0.691	0.699
60	0.03	0	0	0.98	0.94	0.93	0.95	0.91	0.90
80	0.17	0.07	0	1.17	1.07	0.95	1.14	1.04	0.93
100	0.0048	0.049	0.034	1.01	1.05	1.04	0.95	0.95	0.95
120	0.10	0.09	0.07	1.12	1.12	1.11	1.01	1.01	0.93

Table 4: Schedule quality metrics for the sets of Montage workflows executed in CLAVIRE

Table 4 summarizes the results of experiments with CLAVIRE. MDW-C (as opposed to MDW-T and MDW-W) demonstrates the maximum value of the integral criterion for all experiments ($U = 1$ when all deadlines of workflows in a set are met). Values of average reserved time are similar for all three algorithms. Although a part of delayed tasks in the actual schedule is high (from 0.375 to 0.91 for MDW-T), a delay of the entire workflow does not usually exceed 10% from T. Delays of tasks affect delays of workflows mostly for the task-based algorithm. The reason is that it provides more frequent alternation of tasks from different workflows in resource queues, and delay of one workflow can lead to the delays of several others. An analysis of metrics in Table 1 leads us to the conclusion that the clustering-based algorithm is most preferable to use in a distributed environment for two reasons. Firstly, it is more resistant to a violation of a basic schedule, and secondly (as shown in Sections 5.2, 5.3) it provides better schedules than MDW-T and MDW-W.

6 Conclusions

Combining workflows into sets before scheduling helps not only to increase effectiveness of resource utilization, but also contributes to more qualitative schedules. Using additional available information about properties of a scheduling problem can further improve the result. In this work, we consider the problem of scheduling multiple workflows under time restrictions specified by both providers of resources (time windows) and users (soft deadlines). To measure the quality of resulting schedules, we introduce an integral criterion incorporating the level of satisfaction of users' constraints and fairness of distribution of resources. Several types of simulation experiments with synthetic, benchmark and domain-specific data sets precipitate the following conclusions: (i) using the information about time windows (Section 5.1) and soft deadlines (Section 5.2) significantly improves resulting schedules; (ii) algorithm MDW-C outperforms other implementations of the clustering-based approach. The experiments that used the CLAVIRE e-Science infrastructure platform also show that MDW-C is more resistant to a violation of the basic schedule.

However, resulting effectiveness of scheduling depends on many parameters including: (i) number of resources, (ii) maximum level of heterogeneity of a set of resources, (iii) values of communication to computation ratio for different workflows, (iv) shape of directed acyclic graphs of workflows in a set. Further investigations in this area might include studying the influences of these parameters on the quality of schedules, comparing the effectiveness of different clustering schemes, and developing methods to select the most appropriate algorithm depending on the particular formulation of the scheduling problem.

Acknowledgements

This paper is financially supported by Ministry of Education and Science of the Russian Federation, Agreement #14.578.21.0077 (24.11.2014).

References

[1] K. Gupta, M. Singh, Heuristic Based Task Scheduling In Grid, Int. J. Eng. Technol. 4 (2012) 254–260.

[2] S. Kovalchuk, P. Smirnov, K. Knyazkov, A. Zagarskikh, A. Boukhanovsky, Knowledge-based Expressive Technologies within Cloud Computing Environments, in: 8th Int. Conf. Intell. Syst. Knowl. Eng., 2013: pp. 1–10. http://arxiv.org/abs/1312.7688 (accessed March 4, 2014).

[3] H. Topcuoglu, S. Hariri, Task scheduling algorithms for heterogeneous processors, in: Proceedings. Eighth Heterog. Comput. Work., IEEE Comput. Soc, 1999: pp. 3–14. doi:10.1109/HCW.1999.765092.

[4] J. Luo, F. Dong, J. Zhang, Scheduling of Scientific Workflow in Non-dedicated Heterogeneous Multicluster Platform, J. Syst. Softw. 86 (2012) 1806–1818. doi:10.1016/j.jss.2012.10.029.

[5] W. Smith, I. Foster, V. Taylor, Scheduling with advanced reservations, Proc. 14th Int. Parallel Distrib. Process. Symp. IPDPS 2000. (2000).

[6] K. Aida, H. Casanova, Scheduling mixed-parallel applications with advance reservations, Cluster Comput. 12 (2009) 205–220.

[7] H. Zhao, R. Sakellariou, Advance Reservation Policies for Workflows, in: Job Sched. Strateg. Parallel Process., Springer Berlin Heidelberg, Berlin, Heidelberg, n.d.: pp. 47–67. doi:10.1007/978-3-540-71035-6_3.

[8] L.F. Bittencourt, E.R.M. Madeira, Towards the Scheduling of Multiple Workflows on Computational Grids, J. Grid Comput. 8 (2009) 419–441. doi:10.1007/s10723-009-9144-1.

[9] H.-J. Jiang, K.-C. Huang, H.-Y. Chang, D.-S. Gu, P.-J. Shih, Scheduling Concurrent Workflows in HPC Cloud Through Exploiting Schedule Gaps, in: Proc. 11th Int. Conf. Algorithms Archit. Parallel Process. - Vol. Part I, Springer-Verlag, Berlin, Heidelberg, 2011: pp. 282–293. http://dl.acm.org/citation.cfm?id=2075416.2075442.

[10] A. Dukhanov, M. Karpova, K. Bochenina, Design Virtual Learning Labs for Courses in Computational Science with Use of Cloud Computing Technologies, Procedia Comput. Sci. 29 (2014) 2472–2482. doi:10.1016/j.procs.2014.05.231.

[11] K. Bochenina, A Comparative Study of Scheduling Algorithms for the Multiple Deadline-constrained Workflows in Heterogeneous Computing Systems with Time Windows, Procedia Comput. Sci. 29 (2014) 509–522. doi:10.1016/j.procs.2014.05.046.

[12] K. Bochenina, N. Butakov, A. Dukhanov, D. Nasonov, A clustering-based approach to static scheduling of multiple workflows with soft deadlines in heterogeneous distributed systems, in: Procedia Comput. Sci., 2015.

[13] J.Y.J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility Grids using genetic algorithms, 2006 Work. Work. Support Large-Scale Sci. (2006) 1–10.

[14] D.H.J. Epema, M. Naghibzadeh, S. Abrishami, Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds, Futur. Gener. Comput. Syst. 29 (2013) 158–169. doi:10.1016/j.future.2012.05.004.

[15] R. Singh, S. Singh, Score Based Deadline Constrained Workflow Scheduling Algorithm for Cloud Systems, Int. J. Cloud Comput. Serv. Archit. 3 (2013) 31–41.

[16] J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms, Sci. Program. 14 (2006) 217–230.

[17] G. Lu, W. Tan, Y. Sun, Z. Zhang, A. Tang, QoS Constraint Based Workflow Scheduling for Cloud Computing Services, J. Softw. 9 (2014). doi:10.4304/jsw.9.4.926-930.

[18] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds, in: Int. Conf. High Perform. Comput. Networking, Storage Anal., 2012. doi:10.1109/SC.2012.38.

[19] R. Tolosana-Calasanz, J.Á. Bañares, C. Pham, O.F. Rana, Enforcing QoS in scientific workflow systems enacted over Cloud infrastructures, J. Comput. Syst. Sci. 78 (2012) 1300–1315. doi:10.1016/j.jcss.2011.12.015.

[20] H. Arabnejad, J.G. Barbosa, List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table, IEEE Trans. Parallel Distrib. Syst. 25 (2014) 682–694. doi:10.1109/TPDS.2013.57.

[21] U. Boregowda, V. Chakravarthy, Multiple DAG Applications Scheduling on a Cluster of Processors, Comput. Sci. Inf. Technol. (CS IT). (2014) 63–73.

[22] H. Zhao, R. Sakellariou, Scheduling Multiple DAGs onto Heterogeneous Systems, in: Parallel Distrib. Process. Simp. IPDPS'06, IEEE Computer Society, Washington, 2006: p. 14.

[23] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J.L. González-García, T. Röblitz, J.M. Ramírez-Alcaraz, Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid, J. Grid Comput. 10 (2012) 325–346. doi:10.1007/s10723-012-9215-6.

[24] M. Naghibzadeh, S. Abrishami, Deadline-constrained workflow scheduling in software as a service Cloud, Sci. Iran. 19 (2012) 680–689. doi:10.1016/j.scient.2011.11.047.

[25] J. Yu, R. Buyya, C.K. Tham, Cost-based Scheduling of Scientific Workflow Applications on Utility Grids, in: Proc. First IEEE Int. Conf. E-Science Grid Comput., 2005: pp. 140–147.

[26] Y. Yuan, X. Li, Q. Wang, Y. Zhang, Bottom Level Based Heuristic for Workflow Scheduling in Grids, Chinese J. Comput. (2008) 282–290.

[27] F. Suter, DAGGEN: a synthetic task graph generator, (2015). https://github.com/frs69wq/daggen.

[28] Pegasus workflows test set, (2015). https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

[29] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, et al., On the Use of Cloud Computing for Scientific Workflows, 2008 IEEE Fourth Int. Conf. eScience. (2008) 640–645. doi:10.1109/eScience.2008.167.

[30] S. V. Ivanov, S. S. Kosukhin, A. V. Kaluzhnaya, A. V. Boukhanovsky, Simulation-based collaborative decision support for surge floods prevention in St. Petersburg, J. Comput. Sci. 3 (2012) 450–455. doi:10.1016/j.jocs.2012.08.005.

[31] K. V. Knyazkov, S. V. Kovalchuk, T. N. Tchurov, S. V. Maryin, A. V. Boukhanovsky, CLAVIRE: e-Science infrastructure for data-driven computing, J. Comput. Sci. 3 (2012) 504–510. doi:10.1016/j.jocs.2012.08.006.

Biographies

Klavdiya Bochenina (corresponding author, k.bochenina@gmail.com)

is a researcher at eScience Research Institute at ITMO University (Saint-Petersburg). She obtained a Phd in computer science from the same institution with a work on scientific workflow scheduling. Her research activities are focused on several fields including distributed systems, algorithms design and transdisciplinary education.

Nikolay Butakov

is a PhD student at ITMO University (Saint-Petersburg). He got his specialist's Degree in computer science in the Ulyanovsk State University in 2012 with a thesis about discrete filtration algorithms implemented on GPU. His research studies include several aspects in the fields of distributed computation techniques, scheduling and data mining techniques.

Alexander Boukhanovsky

is a director of eScience Research Institute and Professor in High-Performance Computing Department at ITMO University (Saint-Petersburg). His research interests include high-performance computing, computer simulation of complex systems, intelligent computational technologies, statistical analysis and synthesis of spatial-temporal fields, data assimilation in complex systems. He is an author on more than 200 publications, and supervised around 40 R&D projects and 20 PhD theses in these areas. Alexander is a member of program committees of several conferences in the field of computational science, and collaborates as an editor for journals of his research area.

Photo

Klavdiya Bochenina

Nikolay Butakov

Alexander Boukhanovsky