Procedia Computer Science

Volume 80, 2016, Pages 2074–2085

ICCS 2016. The International Conference on Computational Science

Quality-based approach to urgent workflows scheduling

Nikolay Butakov, Denis Nasonov, Andrey Svitenkov, Anton Radice, Alexander Boukhanovsky

ITMO University, Saint-Petersburg, Russia.

alipoov.nb@gmail.com, denis.nasonov@gmail.com, svitenkov@yandex.ru, antonradice@gmail.com

Abstract

Urgent computing capabilities for early warning systems and decision support systems are vital in situations that require execution be completed before a specified deadline. The cost of missing the deadline in such situations can be unacceptable, while providing insufficient results can mean an ineffective solution that may come at a very high cost. In order to provide a solution that is appropriate under the current conditions (i.e. available volume of computational resources, workload, and time available), a new approach is required. In this paper, we present a schema and algorithm of regulating the volume of computations within an urgent workflow to deliver a solution that is as sufficient as possible given the current conditions and deadline. To achieve these goals, we develop an approach that modifies an urgent workflow by changing its structure and the parameters of its individual tasks. Such modifications are based on introducing a notion of quality and applying quality-based models to estimate the sufficiency of solutions generated by the resulting workflow structures. Finally, a special extension of the genetic algorithm that performs quality-based scheduling of urgent workflows is described along with an experimental study to demonstrate its efficacy.

Keywords: Workflow Scheduling, Urgent Computing, Deadline, Genetic Algorithm, Quality

1. Introduction

Nowadays, computational support for emerging critical situations is essential to estimate their development and create an appropriate response. To achieve this computational support, urgent computing techniques are used. Urgent computing assumes the application of special methods to provide the required amount of computational resources to the user and to execute the required computations. The computations themselves are often expressed as a workflow. A workflow is a complex scenario of computations and data processing typically represented by a directed acyclic graph (DAG). Urgent

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.527

computing scenarios additionally require special scheduling techniques in order to be executed efficiently and in time.

These computational situations get more complex as computations can emerge occasionally and unpredictably. As a result, the success of urgent computing significantly depends on the conditions at the moment it is needed. These conditions include the available amount of time for execution, the speed at which the situation is developing, the volume of workload that has been generated, and the amount of available computational resources along with their reliability.

Whereas the existing approaches are dedicated to resource provisioning and phase scheduling, in such conditions it can be very important to apply an appropriate computational scenario to meet the deadline. Inappropriate urgent workflow scheduling may waste available time without any meaningful result. Urgent workflows can also contain one or more critical paths (e.g. sequences of tasks) that cannot be parallelized further and may serve as a bottleneck, decreasing the probability of meeting the deadline and decreasing the reliability of the urgent computing system even further. The cost of this waste can be very high and can result in significant material loss, which in the worst case can mean human lives.

All of the issues mentioned above lead to the necessity of urgent computing methods that can manage not only resource provisioning, computational infrastructure reconfiguration, and workflow scheduling, but also be able to reconfigure the urgent workflow (i. e. change the structure of the workflow) in response to the emerged conditions and to meet the deadline.

To address this problem, a quality-based approach for the organization of urgent computing workflows is proposed in this paper. The term ‘quality’ is specially introduced to describe the usefulness of a workflow (e.g. the degree of precision, count of detected features and points, etc.). By introducing the notion of quality and associated estimation models that can characterize quality depending on various parameters, it is possible to characterize the usefulness of the result of an urgent workflow. Having such models, urgent computing systems can optimize their execution process to deliver optimal quality (e.g. the best results in terms of usefulness) with a guarantee in a limited time, thus reducing the risk of missing the deadline and increasing the reliability of the system. This approach can be used for ad-hoc solution generation as well as developing a set of possible plans in a non-critical period of time. The latter case is especially useful when the set of available resources can vary in time.

This paper contributes the following: (a) a method to describe different workflow applications in terms of quality, thus making it configurable (b) a scheduling method that can exploit quality to create a plan of execution which maximizes the overall quality of workflow and meet the deadline under different conditions.

2. Related works

There are numerous works dedicated to workflow scheduling that attempt to meet either soft or hard deadlines under time constraints. (Abrishami et al, 2013; Bochenina et al. 2014; Mao and Humphrey, 2011; Yeo, 2005; Butazzo, 2008; Visheratin et al., 2015). The goal of these methods is to reduce execution time by optimizing the mapping of workflow tasks and in some situations, the configuration of computational environments. Despite significant improvements, in some cases, these methods fall short if the application is just too big to be successfully executed or if there is a chance of missing the deadline in the emerged conditions. Our approach is targeted to deal with such situations by changing the workflow structure.

Three main approaches to urgent computing can be found in the literature. The first approach (Blanton et al 2012; Yamasaki 2012) assumes the organization and support of private infrastructures like clusters or private clouds. Then, urgent computing applications and infrastructure are specially designed to interact with each other. This approach relies on precise knowledge of the workload volume that may result during the urgent computing phase. This assumption cannot be suitable for all cases. Additionally, the possible time constraint may float or the volume of computational resources may vary.

Furthermore, the cost of private infrastructures can be very expensive and may not be afford in some cases.

The second approach (Beckman et al 2007; Trebon 2011; Krzhizhanovskaya et al 2011) is oriented around organizing capabilities to provide the needed volume of computational resources in time from the first task. The works in the frame of this approach also propose mechanisms to estimate the time spent on the resource provisioning phase. Despite all of the advantages of this approach, it doesn't assume the adaptation of a workflow to the newly emerged time constraints and variability of workload, which can vary greatly in some cases. Also, the obtained resources may be not enough (or they may be unreliable) to reliably execute an urgent workflow as this approach is dedicated to resource provisioning using public clouds and grids that may be unreliable and unstable, and in some cases, provisioning can't even be guaranteed (especially important in the case of grids) due to the possible mandatory behavior of participants who provide resources. The approach can be combined with reliability models of resource provisioning to increase the chance of meeting the deadline and delivering meaningful results.

The third approach (Cencerrado et al 2012; Artés et al 2013) is the closest to ours and is based on managing the volume of computations. This management takes emerging time constraints into account and varies the quality of results by changing parameters in order to reduce the computational workload and meet the deadline. The application of this approach to the prediction of forest fire propagation has been successfully demonstrated. However, this study presented a solution for one particular case that consists of a single computational task and doesn't assume a complex execution scenario like an urgent workflow that is typical in real-time urgent computing applications (Boukhanovsky et al 2012, Krzhizhanovskaya et al. 2011, Balis et al 2013). In contrast, our approach is aimed to deal with quality-based scheduling of urgent workflows

Having mentioned all of the above, it can be concluded that there is no way to manage the quality for workflow-based applications in agile and complex way enough to meet the deadline under arbitrary emerged conditions of the execution.

3. Quality-based extension for urgent computing systems

In order to implement an urgent computing system that can apply quality-based scheduling to efficiently manage workflow structure and thus, a volume of performed computations, the following requirements must be formulated: (1) a description of variable parameters, dependencies, types of packages, etc. and what sub-workflows can replace a task or other sub-workflows; (2) an estimate of the effect of the different variations(parameter and so on) on the results of particular tasks; (3) an estimate of the effect of the different variations on the entire workflow results; (4) a description of an optimized workflow structure whose results improve as time and quality conditions allow.

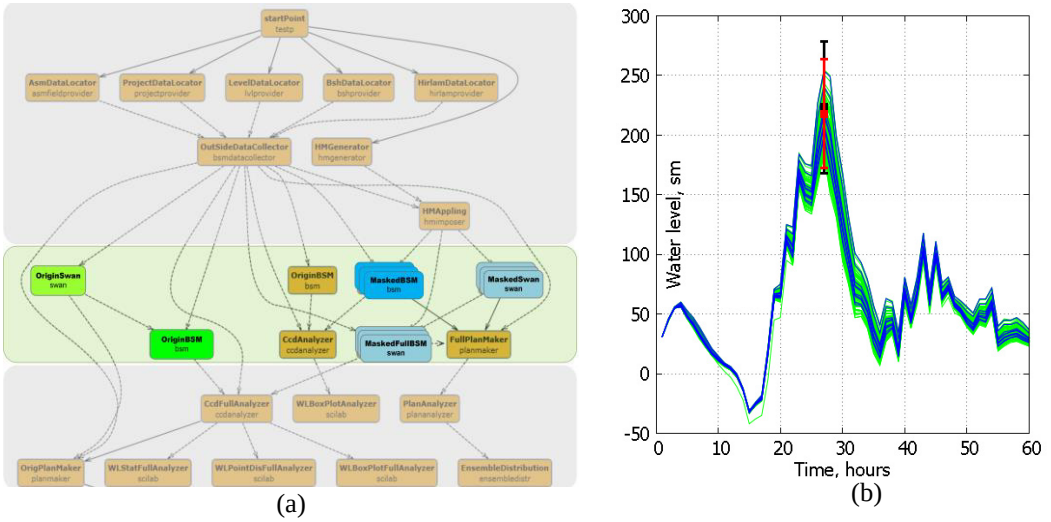
3.1 Models of quality

To meet the requirements stated at the beginning of this section, we must first introduce notions of quality and a quality model of an individual task.

By the term 'quality', we mean the characteristics of a particular tasks result that reflects the usefulness of the results to a user. For example, weather prediction (e.g. values of pressure and temperature) depends on a grid for calculation. If the grid assumes a low-resolution, the predictions of individual points will deviate significantly compared to a grid with a high-resolution. In such situations, it can be said that the quality (or the quality of results) of the task with the high-resolution grid is greater than the task with the low-resolution grid. One of the main features of quality is that it depends on the volume of computations, so different levels of quality will require different amounts of time for completion. It should be noted that in real applications, it is often better to obtain results quicker with reduced quality (Cencerrado et al 2012).

In the case of a workflow, a formulation of quality is more complex. While the quality of a workflow is still determined through its outputs (like for a regular task), these outputs can be influenced by many others tasks even without direct dependences on the tasks which produce the outputs. In this case, a formulation of quality requires us to take into account chains of tasks due to the fact that a workflow is a set of interdependent tasks.

For clarity, several examples are presented below. In figure 3.1.1, a flood uncertainty forecast ensemble is shown. As it is urgent computing task, it is extremely important to provide an estimation of the water level in a certain amount of time.



(a) (b)

Fig 3.1. The flood forecast uncertainty workflow: (a) executed workflow in CLAVIRE; (b) confidence intervals for uncertainty ensemble of 10 and 100 executed water level forecasts

The workflow in the figure is used to estimate forecast uncertainty that may result from input data inaccuracy. In the first block, this inaccuracy is simulated through the applied mask to the HIRLAM weather forecasting data. The second block processes a simulation of the water level prediction. There are two types of quality to hold the computational time in hard deadline restrictions.. The first type is the complexity of computations. Models can be simplified by switching parameters. In the case of the BSM model that generates water level forecast tree main options all available. The SWAN model (light green on the figure), used for wind wave simulation can be reduced and not calculated (BSM parameter "useSWAN" = false). Also, the data of Danish Straits (BSH) from the coastal model can be omitted as well as data assimilation can be switched off. All these parameter states negatively influence on the accuracy (quality) of the BSM forecast but significantly increase its execution time.

On the other hand, workflow execution time can be shortened by reducing the accuracy of uncertainty. In figure 3.1, blue blocks are executed as an ensemble and the amount of blocks can be changed. In fig 3.1.b, the confidence intervals for 10 (blue) and 100 (green) executions are shown. It is clearly seen that the confidence intervals for 100 executions are more precise than the confidence intervals for 10 executions (in this case ~ 18%); however, they take 10 times more computational resources.

Multiscale approaches in physics are a typical example of the propagation of uncertainty (Oden, 2006) and so appearing of sequential dependencies of output results quality. The chaotic behavior of microscale models doesn't imply the direct computation of quantities of interest, but only produces statistical ensembles of states. The uncertainty of microscopic values represents their natural fluctuations and the approximation errors of the model at the same time. The observation of larger numbers or states

provides a higher accuracy of estimations on the average values as well as the dispersion, which can be of interest in practical applications.

Quantum chemistry calculations of the electronic spectrum is another example of a problem where uncertainty is essential. A common model is based on a solution of the configuration interaction problem for the optimal geometry of the molecule (geometry optimization can be done by Orca software being the first step in a workflow). This model obtains the energies and relative intensities (so called strength of oscillators) of electronic transitions so-called zindo and get_spectrum stages). The calculated spectrum is overlaid on the measured spectrum of adsorption of anthracene. The dispersion of adsorption lines can be approximated by a Gaussian function $A(E) = \sum_i f_i \exp[(E - E_j)^2 / 2\sigma^2]$ in vicinity of each transition. However, a certain shift of spectral lines can be observed as well. The reason for both type of variances is related to the heat fluctuations of the molecule geometry and the equilibrium of its electronic state. To account for the heat geometry variance of a molecule (and so to increase precision), the following modification of calculations can be suggested. Instead of the geometry optimization stage, an ensemble of molecular geometries for certain values of temperatures should be provided. The values can be produced by molecular dynamics (MD) simulations or a Monte-Carlo procedure. Pure mechanics or quantum chemistry calculations of total energy are also possible. The former provides a broader set of geometries for the same computational efforts, but a large error in the energy calculation. Since the MD trajectory provides a better searching procedure in the phase space, a combination of MD and quantum chemistry methods can be a reasonable choice (Frenkel, 2001). Anyway, it should be noted that a set of spectral lines will be obtained for each geometry configuration. The resulting smooth function can be found by the same Gaussian approximation as for the case of optimized geometry. This part of uncertainty still corresponds with the computational procedure while the other is essential. The relation between the magnitude of these two types of uncertainty depends on the accuracy of the reproduction of the ensemble as well as the size of the sample set, and is controlled by the given computational time and resources.

The quality of a particular task entirely depends on its semantic and inputs parameters, and may be expressed in the form of a dispersion, expectations matrix from the generated result, or an estimation of the number of found samples in the data. Nevertheless, to be able to manage the workflow structure with a scheduler, it must be expressed in a quantitative way.

In the domain of workflow scheduling, there are task performance models which express the execution times of particular tasks depending on their parameters, assigned computational resources, and internal structure. A task performance model for a task v_k is expressed as an equation $T = F_{v_k}(P_{v_k}, D_{v_k}, R_{v_k})$ where F_{v_k} - is a function of performance that generally depends on: P_{v_k}- package parameters, D_{v_k}- data of the problem, R_{v_k}- resources that are used by the task. In addition to the time of the task, a performance model can be built to estimate data transmission time and special processing of the results (StageIn, StageOut phases). In this case, the performance model takes into account the time required for the transfer of all necessary data on a computing node.

An expression of quality can be formulated in the same way: $q = Q^v(Q_v, P_v)$, where Q_v- set of parameters of v, which have quality estimations that can be used to estimate v(including input data). P_v – set of input parameters which don't have quality estimations (for example, count of iterations of a genetic algorithm), but which influence the quality of (including input data). Q^v- a concrete quality model for the task v. Like for performance model, Q^v expresses the internal structure of the task and determines the quality estimation. It can be an equation, a programming procedure, or a trivial expression that assumes that the task doesn't influence the quality of the output results at all and entirely depends on the quality of input parameters. Additionally, it is better to express quality estimation in a relative way with the help of a convolution:

$$q^{Workflow} = \sum_{i=1}^{k} q_i^{Norm} \omega_i$$

$$q_i^{Norm} = \begin{cases} e^{\frac{|q_i - q_i^{desirable}|}{q_i^{desirable}}}, if\ q_i \geq q_i^{min} \\ e^{q_i}, \quad if\ q_i \geq q_i^{min}\ and\ q_i^{desirable} = 0 \\ -\infty, if\ q_i < q_i^{min} \end{cases}$$

, where q_i- the result of evaluation of the quality of a problem; $q_i^{desirable}$ - the desired quality; q_i^{min} - a limit on the minimum acceptable quality, $q^{Workflow}$ - the quality of the resulting total

This form makes it easy to compare the quality of different output results. Also, the minimal quality can be introduced in the expression above to critically penalize solutions which don't provide minimal quality for each output.

In addition to defining the task quality model and the workflow quality model, it is needed to define what kinds of changes can be applied to a workflow structure and how to apply these changes. We highlight the following basic patterns to represent potential changes to a workflow's structure. Let the workflow $a =< E, V >$, where E set of edges that expresses data dependencies between tasks and V set of tasks.

- **Parameters switching**. Consider some task v in the workflow and an associated quality function for this task $Q_v(P)$, P – is a set of parameters $P = \{p_1, \dots, p_n\}$ whose members correspond to a certain quality of solution. By changing the parameters set the quality can be decreased or increased: $Q_v(P_1) < Q_v(P_2) < Q_v(P_3) < \cdots < Q(P_m)$.
- **Dependency reduction.** Consider two tasks v_a and v_b that are inputs to some other task v, but are considered substitutes (i.e. task v_c can be executed when either one of v_a or v_b have finished executing, or both have finished executing). In the latter case, the associated quality function for task v takes the form $Q_v(p_a, p_b)$, where p_a and p_b are parameters that is provided by the tasks v_a and v_b. Given that v may be executed with one input task (either v_a or v_b), we can rearrange the topology of the workflow by removing a dependency between v *and* v_b (and if task v_b is not necessary anymore, removing v_b too) and estimate the quality with a new model $Q'_v(p_a)$. Thus, this new quality function that can be evaluated with one input parameter rather than two, reducing computational and/or transfer costs.
- **Subworkflow replacing.** Consider some sequence of tasks V_1^{sub} and another sequence of tasks V_2^{sub} that may be substituted for V_1^{sub} . In other words, V_1^{sub} and V_2^{sub} may be swapped in the workflow interchangeably with no failure. Let suppose that V_2^{sub} is a more complex workflow (i.e. more subtasks, data dependencies, etc.) than V_1^{sub} and has a higher quality, which we formally express through the inequality $Q_{V_1^{sub}} < Q_{V_2^{sub}}$, so the subworkflows V_1^{sub} and V_2^{sub} may be interchanged in the workflow when appropriate, increasing or decreasing the quality of the solution. Such transformation of the workflow can be described in a more formal and general way.. Introduce the function: $Valid: A \rightarrow \{True, False\}$. This function takes a workflow*(A - set of workflowss) a*nd returns a Boolean depending on if the structure of the workflow is valid. Next, we define an edge deletion operator as follows: $EdgeDel: A, E \rightarrow A,\ a' =< E', V >,\ E' = (E/e_j): Valid(a') = True$,where e_j- a removed edge. A new workflow structure can be obtained by sequential applications of an operation $ReplaceSubWF$ to the workflow. This operation also takes as arguments set of tasks in the current workflow structure that needs to be replaced and a subworkflow that will replace the chosen tasks. The procedure of subworkflow replacing is finished with the function $EliminateRedundancy$. That function removes redundant tasks and dependencies which doesn't needed for workflow execution, i. e. they doesn't contribute to necessary output results according to the new structure. Then, the ReplaceTask procedure can be formalized as: $ReplaceTask: A, V \rightarrow A,\ a' =< E, V' >,\ V' = (V/v_j): Valid(a') = True$, where v_j is the task to be replaced. Next, $ReplaceSubWF$ is defined as: $ReplaceSubWF: A, A, A \rightarrow A$, $a' =< E', V' >,\ a_{sub} =< E_{sub}, V_{sub} >$, $V' = \left(\frac{V}{V_j}\right) \cup V_{sub},\ V_{V_j}^p = Parent(V_j), V_{V_j}^c = Children(V_j), E_{V_j}^p =$

$\{e_k|\forall e_k \exists v_l \in (V/V_j), v_m \in V_j: e_k = Edge(v_l, v_m)\}, E^c_{V_j} = \{e_k|\forall e_k \exists v_l \in (V/V_j), v_m \in V_j: e_k = Edge(v_m, v_l)\}, E^p_{V_{sub}} = \{e_k|\forall e_k \exists v_l \in (V/V_j), v_m \in V_{sub}: e_k = Edge(v_l, v_m)\}, E^c_{V_{sub}} = \{e_k|\forall e_k \exists v_l \in (V/V_j), v_m \in V_{sub}: e_k = Edge(v_m, v_l)\}, E' = ((E/E^p_{V_j})/E^c_{V_j}) \cup E^p_{V_{sub}} \cup E^c_{V_{sub}}$, where V_j is the newly generated task (i.e. the replaced sub-workflow, which may consist of a single task), a_{sub} is the sub-workflow that replaces task V_j; $Parent: V \to V^p$, $V^p$is the set of all affected tasks, $Children: V \to V^c$, V^c is the set of all dependent tasks, $Edge: V, V \to E$ is a function that returns the edge between vertices, if it exists (in the specified direction), $E^p_{V_j}$, $E^c_{V_j}$ are the set of edges where V_j exists in the original a, and $E^p_{V_{sub}}$, $E^c_{V_{sub}}$ are the set of edges where V_{sub} belongs to a'. Finally, we introduce an operation to remove all imaginary vertices and edge$EliminateRedundancy: A \to A$ edge. Then, any modification of the workflow can be written as a sequence:$(a, V^1_{sub}, V^2_{sub}, V^3_{sub}, \dots, V^n_{sub})$, $n \leq I$, where I is the maximum count of possible changes for A..

- For an urgent computing system operated by a user, is better to introduce this feature in the following way. Let define a notion of *abstract workflow* which describes a set of interdependent tasks (steps) in the form of DAG (directed acyclic graph) that leads to a solution of some problem. The implementation of each task in the workflow can be represented as a subworkflow. Such a subworkflow can consist of a single task and each task of the abstract workflow can have one or more ways to be implemented (e.g. substituted with a subworkflow). So, in this case, the user can define a common strategy for how to solve the problem and provide alternatives that can be swapped by a scheduler of the execution system depending on the current conditions (e. g. time constraints, available resources, etc.) It should be noted that a composition of subworkflows should satisfy mutual dependencies of participating subworkflows.
- **Sequential deadlines.** The workflow structure can be split into a sequence of workflows where each next workflow depends on the previous one and uses its results to improve the quality of the overall result. Consider a workflow that can be represented as $a = a_1 \oplus a_2 \oplus a_3 \oplus \dots \oplus a_n$, where a_1, a_2, a_3, ..., a_n are subgraphs of a, and each one has an associated deadline d_1, d_2, d_3, ..., d_n. It should be noted that some tasks may be duplicated in different workflows. Such a scheme may help to increase the probability of obtaining results with the minimal quality until the hard deadline and continues to improve the results with some period until the hard deadline is finally reached.

In order to use all the above mentioned mechanisms to manage workflow structures, a special scheduling algorithm is required to perform quality-based scheduling.

3.2 Scheduling algorithm

Quality-based scheduling assumes manipulating the workflow structure as well as the schedule of the resulting concrete workflow. A schedule is an ordered mapping of tasks to computational resources where each task has its own time of start. In order to implement quality-based scheduling, a modification of the genetic algorithm is proposed. The chromosome of the algorithm has been extended to include the abstract workflow structure as well as the concrete workflow structure formed by the concrete subworkflows with their ordering and mapping to computational resources.

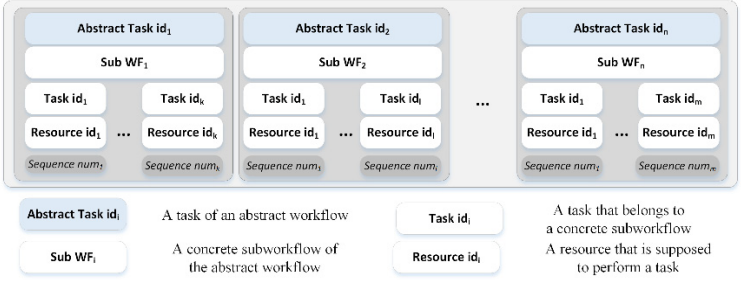


Fig 3.2.1Chromosome structure for quality-based genetic algorithm

a

```
# chromosome of a particular solution
INPUT Individual
OUTPUT mutatedIndividual
num = GENERATE_RANDOM_NUMBER
IF num < mutLimit THAN
    # decide what a mutation type
    # have to be applied
    mut = ROULETTE ReplaceSubWfMut ,
                   SwapMut,
                   ReplaceResourceMut
    IF mut == ReplaceSubWfMut THAN
        # find an abstract task to be replaced
        # with a new subWf
        atask, subWf = CHOOSE_ABSTRACT_TASK Individual
        # create new subwf
        newSubWf = ALTERNATE atask WITH_SUBWF subwf
        mutatedIndividual = REPLACE atask IN Individual
                            WITH newSubWf
        RESTORE_ORDERING mutatedIndividual
        RETURN mutatedIndividual
    IF mut == SwapMut THAN
        task1 = CHOOSE_TASK Individual
        task2 = CHOOSE_TASK Individual
        mutatedIndividual = SWAP task1, task2 IN Individual
        RETURN mutatedIndividual
    IF mut == ReplaceResourceMut THAN
        task = CHOOSE_TASK Individual
        newResource = ALTERNATE task
                      WITH resource
        mutatedIndividual = REPLACE task IN Individual
                            WITH newResource
        RETURN mutatedIndividual
ELSE
    RETURN Individual
# restores the ordering after mutation
RESTORE_ORDERING
INPUT Individual
OUTPUT Individual
tasks = []
FOR atask, subWf IN individual:
    FOR task, resource, priority IN subWf:
        ADD task TO tasks
ordering = []
WHILE tasks is not empty:
    # find all tasks that have
    # all its dependencies resolved
    found_tasks = FIND_TASKS t SUCH (EACH_OF t.parents IN ordering)
                  IN ordering
    # find a task with the greatest value
    #(lesser the value is, more foreground the task is)
    task = CHOOSE t BY MIN(t.priority) IN found_tasks
    REMOVE task FROM tasks
    ADD task TO ordering
FOR task, i IN ordering WITH_INDEX:
    task.priority = i
RETURN Individual
```

b

```
INPUT  parent1, parent2
OUTPUT child1, child2

num = GENERATE_RANDOM_NUMBER

IF num < crossLimit THAN
    p1 = ORDERING_TO_RELATIVE_PRIORITY parent1
    p2 = ORDERING_TO_RELATIVE_PRIORITY parent2
    c1, c2 = SINGLE_POINT_CROSSOVER p1, p2
    child1 = RELATIVE_PRIORITY_TO_ORDERING c1
    child2 = RELATIVE_PRIORITY_TO_ORDERING c2
    RETURN child1, child2
ELSE
    RETURN parent1, parent2

# the function converts absolute sequence number of each task
# to  relative number
# depending on task count in the individual
ORDERING_TO_RELATIVE_PRIORITY
INPUT individual
OUPUT individual

FOR atask, subWf IN individual:
    FOR task, resource, priority IN subWf:
        priority = priority / TASK_COUNT individual

RETURN individual

# the function converts relative number of each task
# to the absolute number
RELATIVE_PRIORITY_TO_ORDERING
INPUT individual
OUTPUT individual

FOR atask, subWf IN individual:
    FOR task, resource, priority IN subWf:
        priority = (priority * TASK_COUNT indvidual)

sorted_tasks = SORT_CONCRETE_TASK individual BY priority
# assign concrete tasks their new indexes
FOR i, (task, resource, priority) IN sorted_tasks WITH_INDEX:
    priority = i
```

Fig 3.2.2 Pseudocode of: (a) mutation procedure; (b) crossover.

Chromosome structure. The chromosome represents a mapping of the abstract workflow structure to concrete subworkflow implementations. An example of the chromosome structure is depicted in Fig 3.2.1. Each abstract task (a task of abstract workflow) has its own concrete implementation as a subworkflow, whose tasks are mapped to computational resources and have a priority that is used to form the task ordering. The ordering is formed for all *concrete* tasks of the resulting workflow structure. It should be noted that priority is used only after a dependency check. In other words, priority is accounted for when choosing the next task for ordering the sequence among the tasks which have all their dependencies already put in the ordering sequence. The ordering and the mapping of resulting workflow tasks is used to build schedule. The detailed description of building procedure can be found in (Rahman et al 2013; Yu and Buyya 2006).

Genetic Operators. The modification of the genetic algorithm includes special mutation and crossover operators. There can be three possible mutation types: (1) replacing the subworkflow of an abstract task; (2) swapping two different tasks in the chromosome in order to change their scheduling priority; (3) replacing a resource where a task will be executed. In the first step, the algorithm decides if it has to apply a mutation, and at the second step, what type of mutation should be applied. The replacement of a subworkflow requires a special treatment of priority fields of a new subworkflow task.

The tasks mentioned above don't have any priority values beforehand and must be assigned them. In that case, all tasks gets the same priority value. This priority value is estimated for the abstract task itself (or its default implementation). Then, priority values for each task (including all subworkflows) is restored as described above. It should be noted that subworkflows often don't contain tasks that can be executed without dependencies of the parent tasks in the abstract workflow.

The crossover operator is a regular single point crossover operator with a minor change. In order to include a subworkflow with concrete tasks into a child, we need to take care of ordering the tasks. The operation of ordering restoration is performed in a similar manner to restoration after mutation. The only difference is that tasks of subworkflows use priority values inherited from their parents.

The pseudocode of mutation and crossover procedures are depicted on Fig. 3.2.2.

The proposed algorithm combines the ability to manage the workflow structure as well as its schedule to solve either a time constrained problem or to build a Pareto front with an application of NSGA-II operators to form the population.

3.3 Architecture of the extended urgent computing system

In order to implement the proposed approach to quality-based scheduling, an extension for CLAVIRE cloud computing platform was made. The architecture of the extended CLAVIRE cloud computing platform (Knyazkov et al 2012) is depicted on Fig. 3.3.1. The platform was extended with an improved 'ExtPackageBase' service, which includes quality models for individual tasks and a 'Control and Planning module' which can calculate a quality estimation to manage workflow structures and an implementation of an algorithm to perform quality-based scheduling.

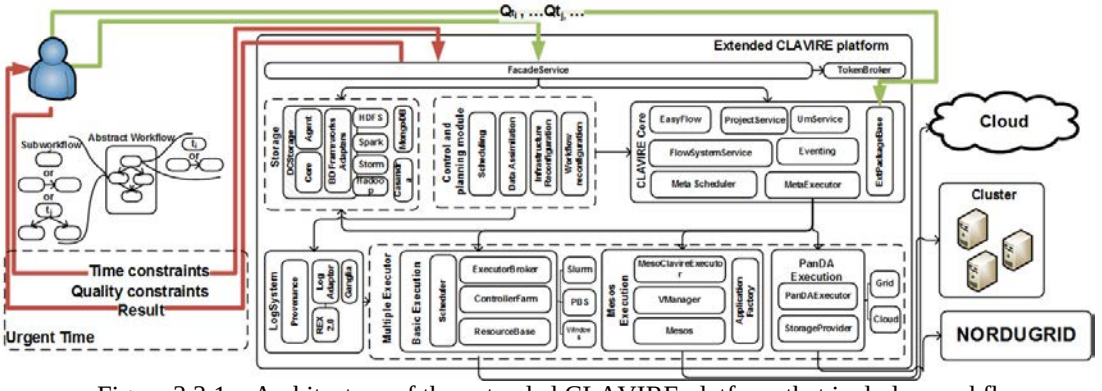


Figure 3.3.1 – Architecture of the extended CLAVIRE platform that includes workflow reconfiguration management (structure management).

The process of dealing with reconfigurable workflows can be described as follows. In the regular time, the user adds quality models for individual tasks either through a web-interface of 'ExtPackageBase' or the CLAVIRE interface adds the required urgent abstract workflow, marking its output parameters as ones which should be managed based on its quality estimations, these estimations user have to provide for each step (or task) in the abstract workflow where it possible. The user has to provide at least one alternative for each step. Optionally, the user can set minimal quality requirements for workflow results and for individual tasks. When a critical situation appears and urgent computing must be applied, the user sets the time and minimal quality constraints and runs the workflow. The rest of the execution process is managed by the extended CLAVIRE system with the help of the proposed approach. The process is depicted in Fig 3.3.1.

The extended platform deals with unstable resources such grids and public clouds as well as private clusters and clouds, while also providing additional chances to meet the deadline. The extended platform also leads to a natural separation of computational workloads in an automatic way: the most critical parts of an urgent workflow are scheduled on reliable and highly controllable private and dedicated resources while computations that can increase the quality of results can be scheduled to unreliable and unstable resources on public clouds, grids, and other such resources. The sequence of deadlines highlighted in the end of section 3.1 is suitable example of critical and noncritical task division. Such

use of computational resources may be further improved with the help of a probability-based approach to reliability estimations for obtainable computational resources introduced in our previous work (Nasonov and Butakov 2014).

4. Experimental Study

To verify our approach, we conducted the following experiment. The flood threat estimation and prevention workflow was chosen for our experimental study. This workflow normally has the structure presented in Fig 3.1.1 (a) and is calculated in the computing environment consisting of three high-performance IBM Blade Server Intel Xeon E7-2830, 128Gb RAM, 1Tb HDD computers. On each server, 3 virtual machines are deployed with the following characteristics: 4 cores, 32 GB. Clusters of 9 virtual machines serves to prevent flooding, described above, and so 9 machines form three racks for 3 machines. These machines are connected with 100 Mbit/sec network channels.

The workflow execution is limited to 3 hours, but depending on the conditions, the available time can be reduced. Also, the same situation is important when only reducing the count of computational resources can be provided or even worse, if some fraction of resources is lost during the execution process as some previously finished results may be useless for the continuation of the computations. In this experimental study, the last situation was reproduced to test the proposed approach. In the process of execution, one machine serves as the coordinator for the entire calculation process. The communication of the coordinator with other machines in its rack much more reliably and faster than with machines in other racks. A loss of connection may occur between the coordinator and one or even two other parts of the cluster. It is necessary to ensure the successful completion of the execution of the process with at least a minimally acceptable quality. In that case, the minimum acceptable quality is a completion of the main branch, i. e. BSM task or SWAN and BSM task.

An extended CLAVIRE platform (with infrastructure manager Mesos) that implements the proposed approach is used to manage the execution process of the workflow and handles failures during the execution and appropriate rescheduling.

During the experiment, failures of resources were generated by disabling one or two racks (like a connection lost) at predetermined moments of the workflow completion - 0%, 10%, 25%, 50%, 75%, 100% of the total completed tasks of the workflow. The value of 100% means absence of connection lost at all, and 0% as the initial absence of connection between the racks.

Fig. 4.1 presents the results of the execution time and the resulting size of the ensemble with and without quality-based rescheduling. The more calculations with a mask (i.e. wider ensemble) is available, the more likely we are to use the safer and cost-effective solution. In this case, it means that the quality assessment also can be made based on the number of completed ensemble elements, i. e. more calculations is finished, the higher the quality is.

From figure 4.1 (a), it is clear that when computational resources are lost, execution time increases dramatically, which can be more obvious in the case of the loss of two racks. Thus, there is a violation of the run-time constraints, marked with the red horizontal line. No violation has only one point - 75% (9-6), due to the fact that the computationally intensive part of work was managed to finish before the violation. Execution time in this case increases non-critically. In that case, there is no need to restructure the workflow.

Reconfiguration of the workflow structure allows a user to get the most needed results without breaking the time constraints, though with some degradation of the quality. As it can be seen, the graph that depicts execution time in case of the reconfigured workflow structure (blue and green line) lies below the limit line. Figure 4.1 (b) shows the size of the ensemble after the reconfiguration. It can be

seen that in case of the failure of two racks, the later the failure occurs, the greater the number of ensemble members will finish before the deadline.

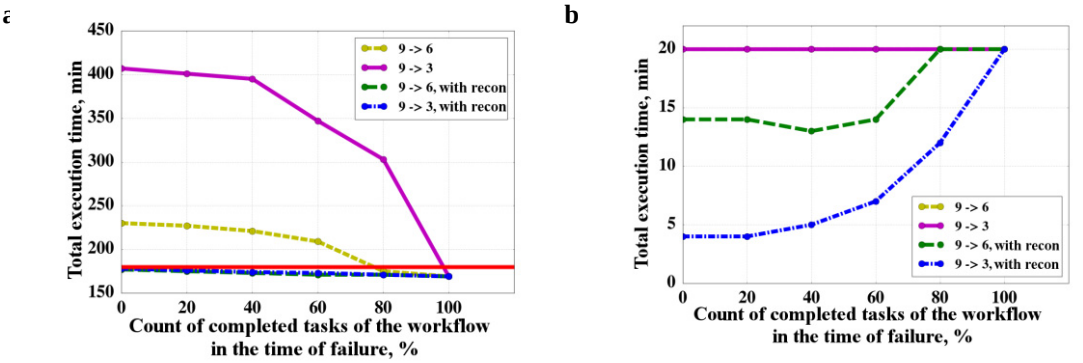


Figure 4.1 - Rescheduling of an urgent workflow execution plan to prevent the threat of flooding in the resource fault conditions in case of quality-based the workflow structure reconfiguration application and without it: (a) total execution time; (б) total size of the ensemble. (***x*** –> *y)* means reduction of available count of computational resources from x to y. '*with recon'* means application of the structure reconfiguration.

5. Conclusion

In this paper, a quality-based approach for urgent computing has been proposed. The approach is based on using models that express the quality of results that can obtained depending on the spent volume of computations, especially for complex scenarios presented as a workflow. This approach provides a scheduler of workflow management system the ability to manage a workflow structure to provide a user the highest quality solution the given time constraints allows. The experimental study shows that this approach can manage to satisfy strict deadline requirements in the case of critical situations, while also maximizing the usefulness of the results. This makes it quite suitable to be applied in managing computations for early warning systems or decision support systems.

In future work, our approach can be extended to account for risks and reliability estimations of using provided computational resources. This may help gain better results in situations when providers of resources are grids or public clouds. The proposed modification of the genetic algorithm has to be extended to an island-based or fork-join modification and parameters of evolution process should be investigated. Also, an investigation of a coevolution approach for quality-based scheduling should be conducted. It can be applied to unify the optimization of the execution plan, workflow structure, and the configuration of the computational environment.

6. Acknowledgements

This paper is financially supported by Ministry of Education and Science of the Russian Federation, Agreement #14.587.21.0024 (18.11.2015). RFMEFI58715X0024

References

Visheratin, A., Melnik, M., Butakov, N., & Nasonov, D. (2015). Hard-deadline Constrained Workflows Scheduling Using Metaheuristic Algorithms. Procedia Computer Science, 66, 506-514.

Abrishami, S. M. (2013). Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. Future Generation Computer Systems 29.1, 158-169

Buttazzo, G. C. (2005). Rate monotonic vs. EDF: judgment day. Real-Time Systems 29.1, 5-26.

Yeo, C. S. (2005). Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. Cluster Computing. IEEE International.

Bochenina, K. (2014). A Comparative Study of Scheduling Algorithms for the Multiple Deadlineconstrained Workflows in Heterogeneous Computing Systems with Time Windows. Procedia Computer Science 29, 509-522.

Mao, M. a. (2011). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM.

Blanton, B., McGee, J., Fleming, J., Kaiser, C., Kaiser, H., Lander, H., ... & Kolar, R. (2012). Urgent computing of storm surge for north carolina's coast.*Procedia Computer Science*, *9*, 1677-1686

Yamasaki, E. (2012). What we can learn from Japan's early earthquake warning system. *Momentum*, *1*(1), 2.

Beckman, P., Nadella, S., Trebon, N., & Beschastnikh, I. (2007). SPRUCE: A system for supporting urgent high-performance computing. In *Grid-Based Problem Solving Environments* (pp. 295-311). Springer US.

Trebon, N. (2011). *Enabling urgent computing within the existing distributed computing infrastructure*. University of Chicago.

Krzhizhanovskaya, V. V., Shirshov, G. S., Melnikova, N. B., Belleman, R. G., Rusadi, F. I., Broekhuijsen, B. J., ... & Pyayt, A. L. (2011). Flood early warning system: design, implementation and computational modules. *Procedia Computer Science*, *4*, 106-115.

Cencerrado, A., Cortés, A., & Margalef, T. A. (2012). On the way of applying urgent computing solutions to forest fire propagation prediction. *Procedia Computer Science*, *9*, 1657-1666.

Artés, T., Cencerrado, A., Cortés, A., & Margalef, T. (2013). Relieving the effects of uncertainty in forest fire spread prediction by hybrid mpi-openmp parallel strategies. *Procedia Computer Science*, *18*, 2278-2287.

Boukhanovsky, A. V., & Ivanov, S. V. (2012). Urgent computing for operational storm surge forecasting in Saint-Petersburg. *Procedia Computer Science*, *9*, 1704-1712.

Balis, B., Bartynski, T., Bubak, M., Dyk, G., Gubala, T., & Kasztelnik, M. (2013, May). A development and execution environment for early warning systems for natural disasters. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on* (pp. 575-582). IEEE.

Rahman, M., Hassan, R., Ranjan, R., & Buyya, R. (2013). Adaptive workflow scheduling for dynamic grid and cloud computing environment. *Concurrency and Computation: Practice and Experience*, *25*(13), 1816-1842.

Yu, J., & Buyya, R. (2006). Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, *14*(3-4), 217-230.

Knyazkov, K. V., Kovalchuk, S. V., Tchurov, T. N., Maryin, S. V., & Boukhanovsky, A. V. (2012). CLAVIRE: e-Science infrastructure for data-driven computing. *Journal of Computational Science*, *3*(6), 504-510.

Nasonov, D., & Butakov, N. (2014). Hybrid Scheduling Algorithm in Early Warning Systems. *Procedia Computer Science*, *29*, 1677-1687.