GRID WORKFLOW DESIGN AND MANAGEMENT SYSTEM

Volodymyr Kazymyr, Olga Prila, Volodymyr Rudyi

***Abstract**: Despite the increased development and use of grid-technologies for solving large-scale computational tasks in various scientific fields, the problem of complexity and efficiency of using distributed grid-resources by end-users, as well as the ensuring the required QoS level is still urgent. The main reasons for this are the complexity of low-level and the absence of full-featured high-level grid tools. The classification of the tasks which are the most suitable for solving in the grid environment is presented in the paper. The issues of supporting different types of tasks at the level of grid middleware as well as at the modern high-level tools are studied in the paper. The paper presents the features of workflow type grid task scheduling taking into account the required parameters of QoS. The results of grid workflow scheduling algorithms research are given in the paper. Based on the analysis the requirements to developed workflow management system and built-in scheduling module have been defined. The results of workflow management system design and implementation are shown. The developed workflow management system can be integrated as an external module to the existing one.*

***Keywords**: grid-environment, workflow, scheduling algorithms, optimization, QoS.*

***ACM Classification Keywords**: C.2.4 Distributed Systems, D.4.1 Process Management.*

Introduction

Currently grid technologies are actively developed and applied to solving of complex high-dimensional problems. But yet there are problems that prevent usage of grid technologies by specialists from different fields which are caused by absence of fully functional grid application development tools and low-level nature of the existing ones. At the same time the process of task adoption to the grid environment is rather complex itself.

The most promising conception in grid computing is workflow which is currently under active development. It provides tools to represent the task as a sequence of subtasks with some certain synchronization schemes. It is a very convenient way of describing a parallel task though currently this type of task representation is almost not supported directly in the mostly used grid middleware. Poor middleware integration does not allow users to utilize the power of grid computing efficiently.

The presence of several parallel blocks in such tasks allows to execute them on different resources for more efficient problem solution. To provide such a solution several factors should be taken into account and the most important of them is the expenses for data exchange between utilized resources. Workflow scheduling is a complex problem and it often cannot be solved with the help of the traditional methods that are used for scheduling single tasks in grids. That is why a separate class of algorithm, workflow scheduling algorithms, was introduced to solve the problem. It is often not just one particular algorithm, but a set of methods which are applied basing on the determined workflow type.

Meanwhile there are many workflow management systems (WMS) that work either as separate tools or are integrated in the existing frameworks that provide high-level tools for creating grid portals and designing workflows (e.g. Taverna, WS-pgrade). But most of them do not solve the problem of efficient workflow scheduling.

Another component that is very important for grid end-users is the ability of a grid system to provide its consumers with the required quality of service (QoS). It results in the need to allocate task on a set of resources which is most suitable for its execution depending on the QoS information provided by the resources. While for non-commercial community grids it is limited to estimate completion time (ECT), commercial utility grids can also operate with costs of calculations and some other parameters. This information should be used not only to determine which resources are able to execute some task, but also to find out which of them will be able to perform it in the minimal possible time with the minimal expenses. Apart from this in the case of strict completion time requirements it is important that the environment can guarantee the completion of such a task in determined time. Thus tools for monitoring execution and failover mechanisms are needed.

The paper is dedicated to the research of methods of scheduling the execution of workflow in the grid environment and overview of the existing grid workflow management tools. The requirements to the scheduling module of WMS and the results of WMS design and development on the basis of QoS-based scheduling algorithm are provided in the paper.

Grid Tasks' Type Classification

There exists a well-known classification of tasks. There is a subset of tasks that are most suitable for solving in the grid environment. It can be divided into three types. This determines the similarity of infrastructures of the VOs, created for different branches of science. Let us consider the types of tasks which are calculated in the grid-environment.

1. The first one is the task of a great dimension; its implementation may be characterized by internal parallelism, though calculating does not require the usage of the whole grid-infrastructure available. Therefore the necessity of the grid-environment usage adds up to the usage of a high-end remote cluster resource. The effectiveness of the cluster resource usage may be significantly reduced because of computational data transmission losses. From the point of view of a grid-portal's developer the tasks of this type are the easiest, the effectiveness of their calculating depends on the effectiveness of the software support of the task itself. Similar tasks will be called as those, which present a common computing unit.
2. The task is a common computing unit, which is decomposed into several subtasks by means of data distributing into several portions for parallel processing. This programming model is called Single Program Multi Data [Voevodin, 2004]. Such tasks will be referred to as the ones which are characterized by the data parallelism.
3. The task is decomposed into several parallel-sequential subtasks with a certain scheme of computing synchronization. This type of tasks is known under the term grid workflow. Such a type of tasks may be complicated by the periodical synchronizations between partially parallel calculation blocks.

The complex type of the task which includes the characteristics of several types should be considered separately. There may also occur a particular case of type three when the task is decomposed into totally independent parallel subtasks. In the latter case the process of adaptation to grid is significantly simplified due to the absence of necessity of synchronization between the calculation blocks.

Grid Middleware and High-Level Tools Support Issues for Different Types of Tasks

Consider the support of the above types of tasks by the ARC Nordugrid (http://www.nordugrid.org/) and gLite grid middleware (http://glite.cern.ch/), included as one of the major middleware providers in EMI (http://www.eu-emi.eu/), and the most used in the Ukrainian National Grid Infrastructure.

ARC supports the following two basic specifications for job description JSDL [GFD-R.136, 2008] and xRSL [NORDUGRID-MANUAL-4, 2013]. For the purposes of integration with gLite-middleware it supports JDL [WMS-JDL, 2011]. GLite uses the JDL specification.

As mentioned in the documentation, JSDL is a language for describing the submission requirements of individual jobs. It does not attempt to address the entire lifecycle of a job or the relationship between individual jobs. External language to describe parametric jobs, co-allocation, workflow can be JSDL-aware and can refer to the JSDL description of single jobs for each node in the activity. The definition of such a language should be a separate and independent effort that can build on top of a standard such as JSDL and is therefore outside the scope of JSDL. So, only the first task type is purely supported by JSDL.

xRSL specification also supports just the first job type. The possibility of the automated submit of a set of tasks is not supported directly.

The Job Description Language is a high-level, user-oriented language for describing jobs and aggregates of jobs such as Direct Acyclic Graphs (DAG) and Collections. JDL introduces the concept of type of computing unit - Job, DAG and Collection and the concept of job type as Normal and Parametric.

A DAG represents a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs (figure 1). The jobs are nodes (vertices) in the graph and the edges (arcs) identify the dependencies.

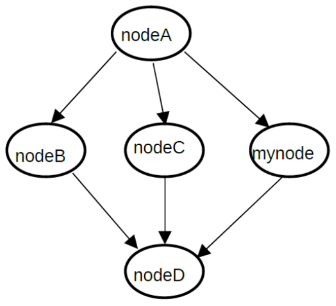


Figure 1. JDL representation for a DAG

DAG differs significantly from the one for a job and it is for sure more complex.

A parametric job is a job having one or more parametric attributes in the JDL. The parametric attributes vary their values according to another attribute also specified in the job description. The submission of a parametric job results in the submission of a set of jobs having the same descriptions apart from the values of the parametric attributes. So the defined second type of task may be specified by the parametric JDL job.

A job Collection is a set of independent jobs that for some reasons have to be submitted, monitored and controlled as a single request.

We can conclude that JDL specification supports three types of tasks in the most complete way. But it also has certain limitations – it does not allow to specify periodical synchronization between units and to define arcs' weights.

The problem of standardization of the second and third type's specification for different middleware remains open; its solving is expected within EMI. It enables us to conclude that the mechanism of planning and automatic submitting of workflow to grid must be laid over the middleware level.

Lately there has been a rapid development of high-level workflow management systems (WMS). WMS provide an infrastructure to setup, execute, and monitor scientific workflows. The notions workflow system for business processes management not adapted for grid environment and grid workflow systems are often mixed. The former possesses a good functional interface for workflow designing, still not oriented to the usage of the distributed grid-environment. Though some of them provide mechanisms of integration with the distributed grid-environment, they are too complicated from the realization point of view.

Existent solutions of workflow designing are either separate systems of designing with their own methods of authentication (Taverna [Hull, 2006], Montage [Jacob, 2009], Triana [Taylor, 2005], Kepler [Ludäscher, 2006], WS-VLAM [Korkhov, 2010], MathCloud [Voloshinov, 2010], MaWo [Sukhoroslov, 2010]) or built into frameworks tools for grid-portals designing (GridNNN [Shamardin, 2010], WS-PGRADE portal [Farkas, 2010], Lunarc app [Appleton, 2010]). Most of the analyzed tools intend deployed web-services on remote resources, used for identifying workflow stages; they cannot identify the task block as a binary executable. Some of them are specialized for certain branches of scientific tasks, i.e. MathCloud, WS-VLAM. The comparative analysis of the systems is given in the works [Petrenko, 2011] та [Curcin, 2008]. Among the analysed WMS one must single out the framework WS-PGRADE portal, and the independent workflow management system Taverna as the most functional and currently supported. Yet, none of the systems analysed solves the problems of effective workflow tasks scheduling taking into account the Quality of Service (QoS) requirements, demanded by a user.

In our opinion the important requirement to the workflow management system is the internal metabroker or the integration with the external systems of workflow scheduling which enables the user to choose the required level of QoS. Due to the fact that the grid middleware brokers are not specialized for the workflow task type the problems of workflow scheduling are to be solved at the level of metascheduler. The effective workflow scheduling algorithms are at the stage of working out.

The Formalization of the Task of Grid Workflow Execution Optimization

A parallel application is generally modeled by a precedence-constrained task graph, which is a directed acyclic graph with node and edge weights (the nodes represent the tasks and the directed edges represent the execution dependencies as well as the amount of communication). In this model a task cannot start execution before all of its parents have finished their execution and sent all of the messages to the machine assigned to that task [Forti, 2006].

In figure 2 the workflow task structure example is presented. Under the unit we mean a single grid-task, JobType=Normal.

For effective workflow scheduling the following parameters must be defined in addition to the standard unit parameters:

$$\{ECT, Memory,\{T\}\},$$

where ECT - estimated completion time;

Memory – memory requirements;

{T} – set of links to other modules (one-way communication between nodes).

Each relationship is defined by the data capacity parameter – the amount of data transferred between the units.

To simplify the structure the memory requirements are omitted. Estimated computation time is expressed in abstract measure of K.

It is worth noting that JDL representation for a DAG does not provide the ability to specify weight of the connections between the units.

For each node of the network structure the following parameters must be defined (figure 3):

$$\{CPU, Memory, QueueSize, Cost,\{R\}\},$$

where CPU – CPU power;

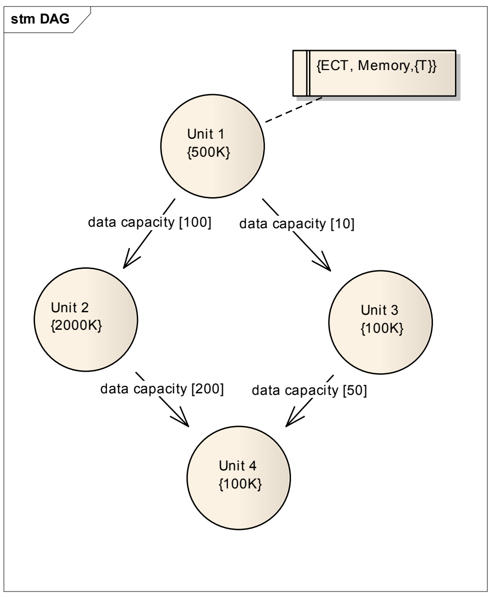
Memory – available memory;

QueueSize – the size of the task queue;

Cost – cost calculations on this cluster;

{R} – multiple links to other network nodes (bi-directional communication between the nodes).

Each relationship is defined by the bandwidth parameter.



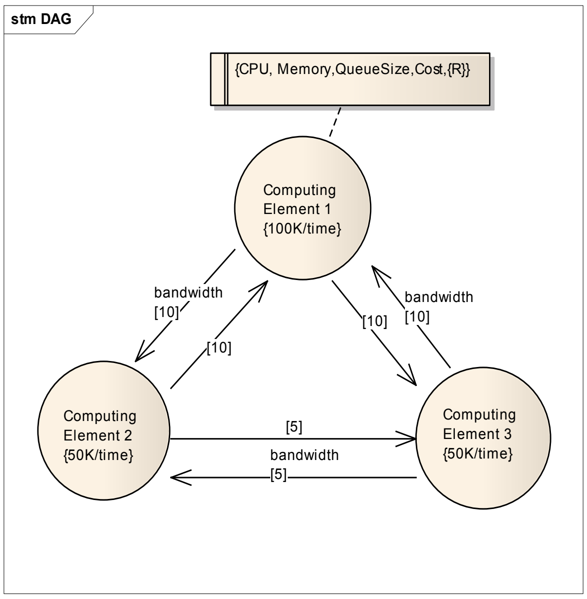

stm DAG
{CPU, Memory,QueueSize,Cost,{R}}
Computing Element 1 {100K/time}
bandwidth [10]
[10]
[10]
bandwidth [10]
Computing Element 2 {50K/time}
[5]
bandwidth
[5]
Computing Element 3 {50K/time}

Figure 2. Grid workflow structure example **Figure 3.** Grid network structure

In figure 3 to simplify the structure the memory, cost and size of the queue parameters will be omitted. CPU power is specified in the abstract measure K/time.

Cycles and branches support requirements to the specification language were not considered because such relationships are rarely found among large blocks of grid workflow real tasks. Furthermore, the presence of such branching is not justified from the standpoint of efficient use of the grid environment in such an embodiment.

The economic risk analysis task description is presented in Figure 4.

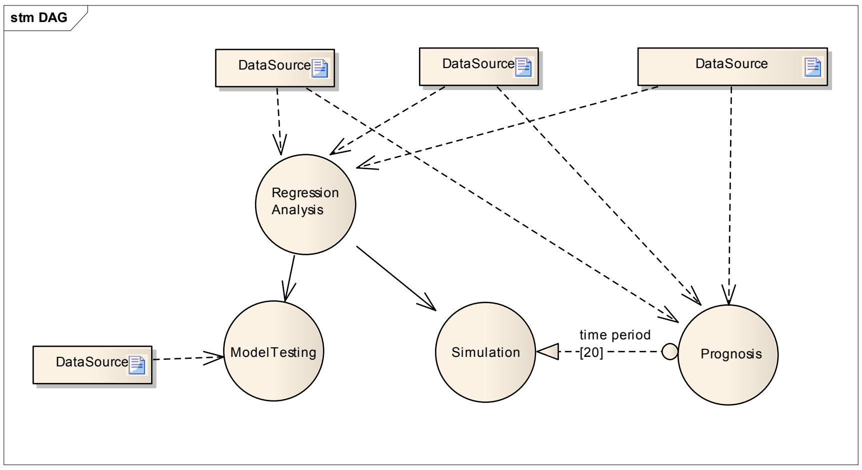


Figure 4. Economic risk analysis task description

Communication between the modules simulation and prognosis is periodic synchronization. It means that simulation can be started as soon as the partial set of values for the factors was produced at prognosis stage.

The objective function is:

$$\varphi = f\,(\text{time, cost}) \rightarrow \min, \tag{1}$$

where time – total running time of workflow;

cost – the total price of the use of resources.

The view of the objective function depends on the parameters of the requested QoS – time or cost priority.

Since the dependencies between tasks can be quite complex, there is the problem of effective scheduling tasks presented in the form of a workflow. Traditional methods of scheduling in grid systems cannot always provide effective planning of this type of problems. This led to the fact that there was a separate class of scheduling algorithms for such tasks called grid workflow scheduling algorithms.

These algorithms must take into account:

- heterogeneous nature of Grid systems;
- the presence of a large number of users competing for shared resources;
- the cost of communication between tasks.

Scheduling of precedence-constrained task graphs is an NP-complete problem in its general form [Forti, 2006].

Grid Workflow Scheduling Algorithms

Scheduling algorithms for workflows can be divided into two categories – with effort-based and budget-based constraints. The first category is more common for community grids where costs are not considered during schedule optimization, while the second one is the most popular for utility grids with service-based access.

Best effort-based scheduling methods. In general these methods are based on some kind of heuristic that allows to get acceptable suboptimal solution within affordable time frame [Topcuoglu, 2002]. Not all the methods were separately developed for workflows scheduling, in fact some of them just allow to execute some set of depended tasks, but in some cases this is enough to create a suboptimal schedule.

One of the simplest algorithms that is applied to workflow scheduling is *Myopic* algorithm. It is a single-task scheduling method, so it does not take into account the nature of workflows. The main idea of the algorithm is to send every next subtask that is ready for execution to the resource that will be able to finish it in the minimal time. It is very easy in implementation, but has some flows, e.g. data exchange between subtasks is ignored.

Min-Min algorithm is based on calculating minimal completion time (MCT) at all resources for all workflow parts. Each subtask allocation has a priority based on MCT. The less is the value, the higher is the priority. Min-Min method shows good results in the grid environments with a big amount of long-running subtasks. While it has performance tradeoffs in case if all subtasks have nearly the same little MCT. *Min-Max* algorithm is pretty similar to Min-Min, but priority assignment is inverted – the bigger is MCT, the higher is the priority. It shows good results for environments with prevailing short tasks.

Sufferage algorithm incorporates implicit analysis of the grid environment heterogeneity. It is implemented by simple, but rather effective approach. For one given task two possible values of MCT are computed using two resources – MCT1 and MCT2. By subtracting these two values it is possible to estimate the difference between used resources. Based on this difference priorities for subtasks' allocations are assigned [Sulistio, 2004].

Another heuristic algorithm that was specifically developed for heterogeneous grid environments is *HEFT* (Heterogeneous Earliest Finish Time). Actually it can be considered as the most popular heuristic that is used separately or as a basis for other heuristics. It is based on creating ranks for allocations on the basis of recursive calculation of average execution and data exchange time including parent subtasks.

Genetic algorithms allow to perform optimization in some limited search space. This approach is suitable for grid environments [Zomaya, 1999] as there is no complete information about the current state of the network and its resources. By choosing the appropriate fitness function it is possible to get acceptable solutions for the scheduling problem. Another advantage of the method is its ability to implement non-stop evolution during the whole planning process, so the workflow schedule is being optimized while it stays in the execution queue.

Solutions are represented as so-called encoding strings which can be of several types. The most popular approach is to use a pair of two strings – the matching string and the scheduling one or the two-dimension scheduling string. The matching string defines which subtask should be sent to what resource, while the scheduling one defines the order of subtask execution. The two-dimensional scheduling string contains rows with the available resources and columns with the information about subtask execution order on these resources.

Simulated Annealing is an algorithm that is based on Monte-Carlo method and modeling a real-life process during the schedule [Feo, 1995]. By randomly generating solutions using Boltzmann distribution it mimics process of materials annealing using different temperatures to make their crystal structure more consistent. The algorithm has three steps – generating random schedule, evaluating its efficiency, e.g. by estimated completion time (ECT) and decreasing “temperature” value. For each “degree” a certain number of schedules is generated. Each time generated schedule is better than the current one, it becomes a current solution and steps are repeated again until the temperature reaches some predefined value.

Budget-constrained methods are especially relevant to commercial grids [Tsiakkouri, 2005] where besides execution time users want to minimize their expenses of computations. This introduces the need of optimizing two opposite values as in general less execution time means more powerful and more expensive resources. To do so QoS is used. It allows a scheduler to estimate workflow execution time and determine its needs, e.g. access to some information sources, memory and storage amount etc. While in community grids tasks can have some QoS-requirements too, no one can guarantee that they can be satisfied.

Currently QoS-based scheduling is in the early development state and most systems allow only two type of constraints that are applied to tasks – *deadline-based* and *cost-based* scheduling.

There are two generic methods of deadline-based scheduling – the *deadline distribution* algorithm and a *backtracking method*. The deadline distribution method is based on the idea that there are special subtasks which are called synchronization blocks. These subtasks have several parent blocks and to execute such kind of a subtask a scheduler has to wait until all of parent subtasks are completed (figure 5).

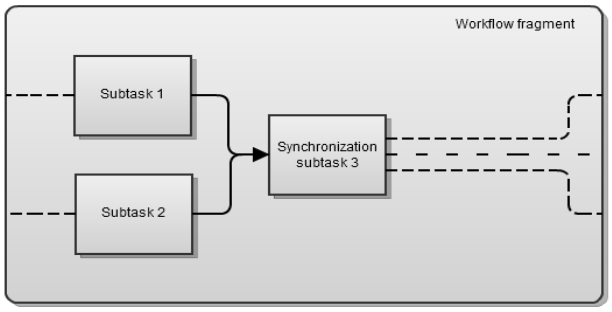


Figure 5. Synchronization subtask example

For all paths between synchronization blocks estimated completion time is executed and deadline are built. If total execution time allows to complete the task before the deadline then such a schedule is considered optimal enough, in the opposite case another schedule variant is checked.

The backtracking algorithm is recursive and uses checks of total execution time on each step. If it exceeds value, defined by the deadline, then the current schedule step is canceled, the cheapest resource is deleted from the resources list and the next variant is checked. Algorithm steps are repeated until all blocks are allocated respecting the given deadline.

Cost-based methods have minimal expenses as the primary target, while completion time still should be as little as possible. One of such methods is *LOSS and GAIN* method. The idea of the algorithm is rather simple and elegant. At first computation expenses are calculated. If they are lower than some predefined value, it means execution time can be decreased in bounds of a specified budget. In this case GAIN operation is performed. GAIN operation means “to gain the best ECT with the minimal cost’. This operation is repeated until budget limit is reached.

In case the expenses of a current schedule are higher than the predefined amount, LOSS operation is applied. This operation means “to reduce expenses as much as possible with the minimal loss of ECT”. This operation is performed until expenses meet the budget constraints.

The research of existing algorithms showed that “pure” methods are not always able to provide satisfying solution and hybrid or “metaheuristic” solutions are considered to be most effective. The latter does not create an optimal schedule for workflows of a specific type, but basing on the determined task type it decides which heuristic will be better for the given task.

Though such an approach drastically increases the scheduling quality it has a drawback of time consuming schedule for each task. Apart from this a user can have preferences between minimal completion time and minimal expenses. The designed algorithms should not just allow to select automatically the used strategy but also to choose between fast and optimal scheduling.

QoS-based scheduling algorithm

The task structure presented in Figure 3 can be divided into the following levels that are determined by parallel blocks: Level 1 - Unit 1, Level 2 - Unit 2, Unit 3, Level 3 - Unit 4.

Optimal solution contains optimal solutions at each level, which means that the task has the property of optimality [Cormen, 2009].

The flowchart of the algorithm is shown in Figure 6.

Consider the example of the algorithm for the workflow shown in Figure 2 and the network structure shown in Figure 3.

For the simplicity we take the objective function as

$$f\,(\text{time}) \rightarrow \min, \qquad (2)$$

The input data is represented in a table structure (table 1-2).

Step 1. Determined levels:

Level 1: Unit 1

Level 2: Unit 2, Unit 3

Level 3: Unit 4.

Step 2. For set №1 define all possible variants of unit allocation:

U1CE1, U1CE2, U1CE3.

Step 3. Determine the value of the objective function for each variant.

U1CE1: 500k / 100k = 5;

U1CE2: 500k / 50k = 10;

U1CE3: 500k / 50k = 10;

Step 4. Save the value of the objective function for each variant.

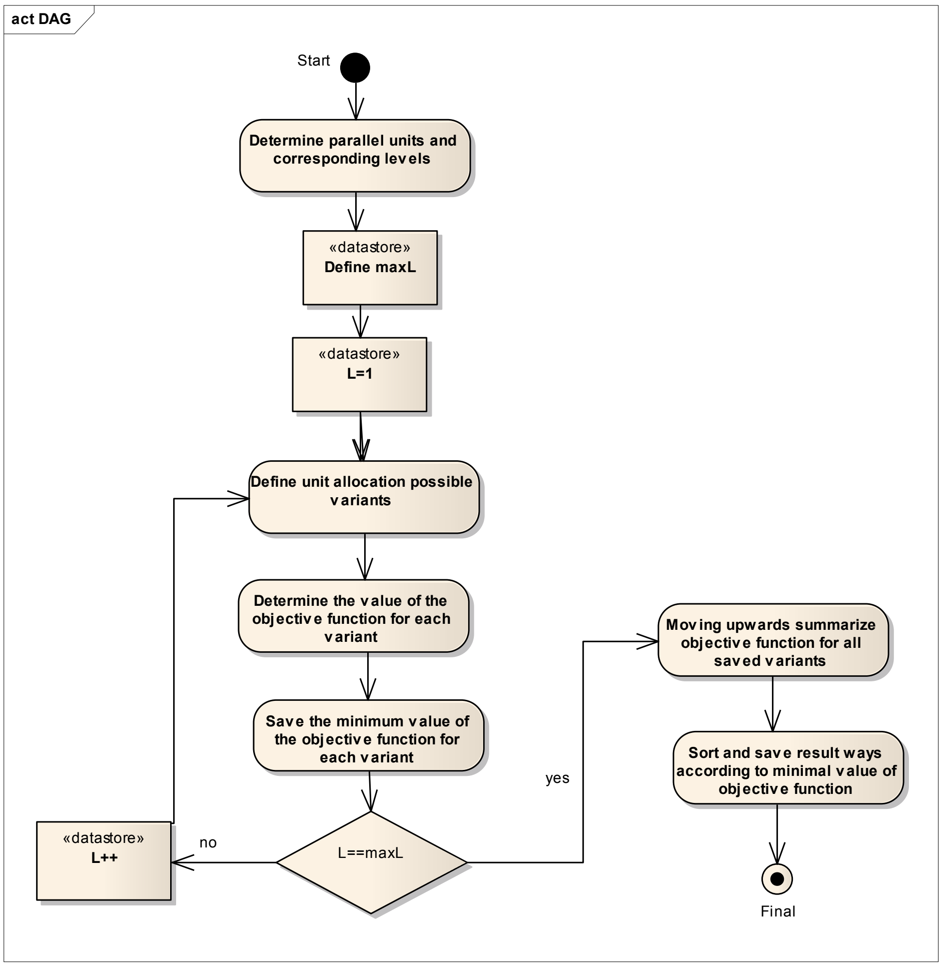

act DAG
Start
Determine parallel units and corresponding levels
«datastore» Define maxL
«datastore» L=1
Define unit allocation possible variants
Determine the value of the objective function for each variant
Save the minimum value of the objective function for each variant
L==maxL
no
«datastore» L++
yes
Moving upwards summarize objective function for all saved variants
Sort and save result ways according to minimal value of objective function
Final

Figure 6. The QoS-based scheduling algorithm flowchart

Table 1. Workflow structure

	1	2	3	4	ECT
1	#	100	10	#	500k
2	#	#	#	200	2000k
3	#	#	#	50	100k
4	#	#	#	#	100k

Table 2. Grid network structure

	0	1	2	CPU
0	0	10	10	100k/time
1	10	0	5	50k/time
2	10	5	0	50k/time

Step 5. Turn to set № 2. Repeat steps 2-4. First, we define the computational cost for U2 and U3 (for ease of computation), and then for all possible combinations of the location of the previous set of blocks.

a) U2CE1U3CE2: 2000k/100k + 100k/50k = 22;

b) U2CE1U3CE3: 2000k/100k + 100k/50k = 22;

c) U2CE2U3CE1: 2000k/50k + 100k/100k = 41;

d) U2CE2U3CE3: 2000k/50k + 100k/50k = 42;

e) U2CE3U3CE1: 2000k/50k + 100k/100k = 41;

f) U2CE3U3CE2: 2000k/50k + 100k/50k = 42;

The calculated objective function for each combination of set №2 is presented at table 3.

Table 3. The objective function value for each combination of set №2

	U1CE1 (5)	U1CE2 (10)	U1CE3 (10)	Min
U2CE1U3CE2 (22)	22+5+0+10/10	22+10+100/10+0	22+10+100/10+10/5	28(U1CE1)
U2CE1U3CE3 (22)	22+5+0+10/10	22+10+100/10+10/5	22+10+100/10+0	28(U1CE1)
U2CE2U3CE1 (41)	41+5+100/10+0	41+10+0+10/10	41+10+100/5+10/10	52(U1CE2)
U2CE2U3CE3 (42)	42+5+100/10+10/10	42+10+0+10/5	42+10+100/5+0	54(U1CE2)
U2CE3U3CE1 (41)	41+5+100/10+0	41+10+100/5+10/10	41+10+0+10/10	52(U1CE3)
U2CE3U3CE2 (41)	42+5+100/10+10/10	42+10+100/5+0	42+10+0+10/5	54(U1CE3)

Step 6. Turn to set № 2. Repeat steps 2-4.

U4CE1: 100k / 100k = 1;

U4CE2: 100k / 50k = 2;

U4CE3: 100k / 50k = 2;

In table 4 we present only the results of calculations.

Table 4. The objective function value for each combination of set №3

	a (28)	b (28)	c (52)	d (54)	e (52)	f (54)	Min()
U4CE1 (1)	28+1+0+50/10	28+1+0+50/5	52+1+200/10+0	54+1+200/10+50/5	52+1+200/10+0	54+1+200/10+50/10	34(a)
U4CE2 (2)	28+2+200/10+0	28+2+200/10+50/5	52+2+0+50/10	54+2+0+50/5	52+2+200/5+50/10	54+2+200/5+0	59(в)
U4CE3 (2)	28+2+200/10+50/5	28+2+200/10+0	52+2+200/5+50/10	54+2+200/5+0	52+2+0+50/10	54+2+0+50/5	59(д)

Step 7. Select the minimum value of objective function for block 3. According to calculations min *f*(x) = 34 for (U4CE1, U2CE1U3CE2, U1CE1).

It should be noted that for more complex kinds of relationships the algorithm requires additional conditions. For example, if a task has the structure as shown in Figure 7, it is necessary to evaluate the estimated time of delivery of data from Unit 1 to Unit 4 corresponding to the run-time Unit 3. If the execution time is longer, this relationship can be ignored.

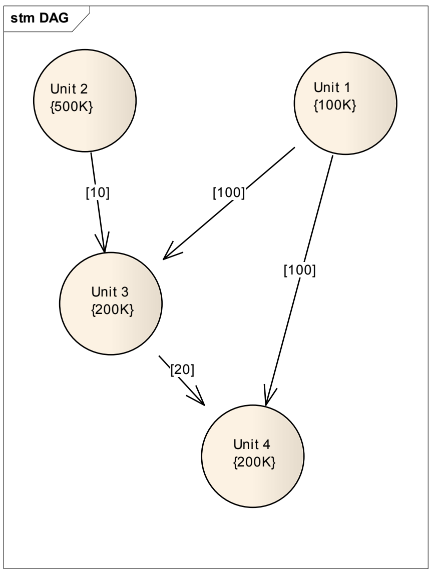
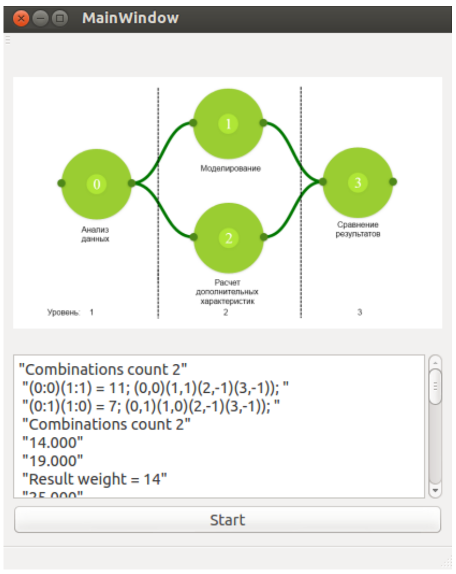


Figure 7. Special type of workflow



```
"Combinations count 2"
 "(0:0)(1:1) = 11; (0,0)(1,1)(2,-1)(3,-1)); "
 "(0:1)(1:0) = 7; (0,1)(1,0)(2,-1)(3,-1)); "
 "Combinations count 2"
 "14.000"
 "19.000"
 "Result weight = 14"
```

Figure 8. Schedule module application

Getting and saving on the last step of the algorithm sorted several task allocation variants allows to make dynamic rescheduling of the task in case of failure of the resources selected for placement of the unit.

The algorithm was implemented in C + +. Figure 8 is a screenshot of the qt-application to determine the best allocation case for the upper specified task and network structures.

Grid Workflow Management System Design

Before starting the actual design of grid workflow management systems authors highlighted a set of features that are in their opinion essential for such a type of software as they are not supported in most workflow management systems or have some limited support.

1. EMI Standard compliance

To be able to use different underlying middleware Workflow Management system should use some common interface to perform requests to low level components. EMI is the de-facto standard provider for grid middleware. Currently NorduGrid ARC, gLite, UNICORE and dCache are considered as EMI-compliant.

2. Control points mechanism

For efficient scheduling the system should know if some subtasks were completed with errors in order to reschedule such parts.

3. Choosing a scheduler algorithm based on the workflow type

Since algorithms can show various scheduling quality on different workflow types, the designed system should allow to determine the task type automatically and to decide which algorithm should be used.

4. Sometimes the user may want to assign a particular subtask of workflow to the predefined set of resources, e.g. because of security measures. The workflow management system should allow to “pin” subtasks to resources and perform optimization on the “not pinned” resources and subtasks.
5. The possibility to choose the Runtime Environment for the task and its subtasks.

Since the blocks of the workflow may have different requirements to the execution environment, as well as a list of available software, it is important to be able to select an execution environment for each unit or group of units separately.

6. The ability of dynamic redistributing of tasks units during execution

Possessing a mechanism of control points one can achieve a more effective scheduling in the case of the subtasks completion failure. In such a case, it is possible not to change the existing allocation, but to move the unit from the failed resource to any available. Herewith time and cost loss may happen, but at the same time there is no need to make rescheduling for the already scheduled tasks, which is beneficial to the overall execution time and the total cost of the task.

7. Support for periodic exchange of data between the partially parallel blocks

There may be situations when the unit can start work as soon as there is prepared portion of the data from the other parallel block. Further units are operated in parallel, periodically synchronizing. The developing tool should provide the necessary synchronization primitives and messaging facilities.

8. Ability to define service level requirements (taking into account QoS)

For tasks limited by time and cost of performing it is important to have the opportunity to assess these values before getting started. In such a case, you must obtain the agreement of the level of resources provided by the service provider. In the case of public grids such information is usually not available. Therefore, a module that would allow to obtain statistics on the basis of the already completed tasks and analyze QoS is needed.

9. Duplicating blocks

In the case of high run-time requirements, the most time-consuming units of the task can be duplicated on multiple resources. This approach will increase the cost parameter value, but decrease the probability of the task failure and time delays in case of resources failure.

Architecture of WMS is shown in figure 9.

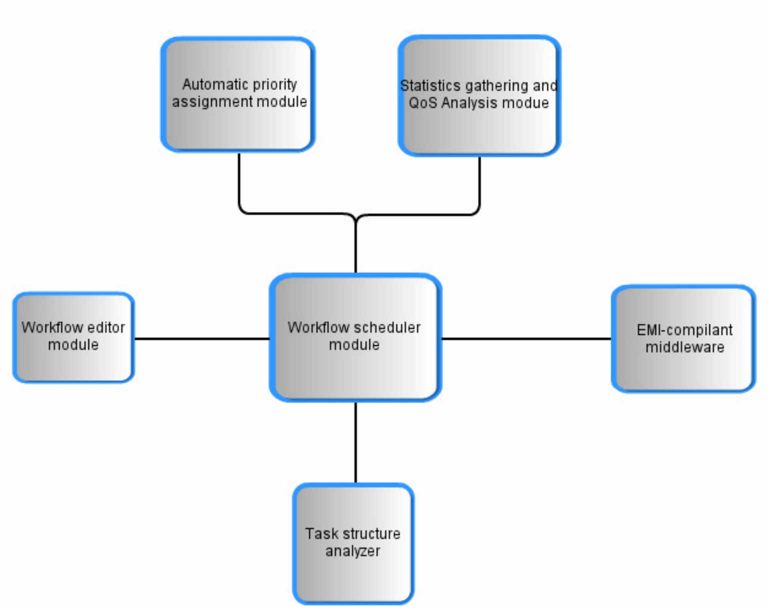

Automatic priority assignment module
Statistics gathering and QoS Analysis modue
Workflow editor module
Workflow scheduler module
EMI-compilant middleware
Task structure analyzer

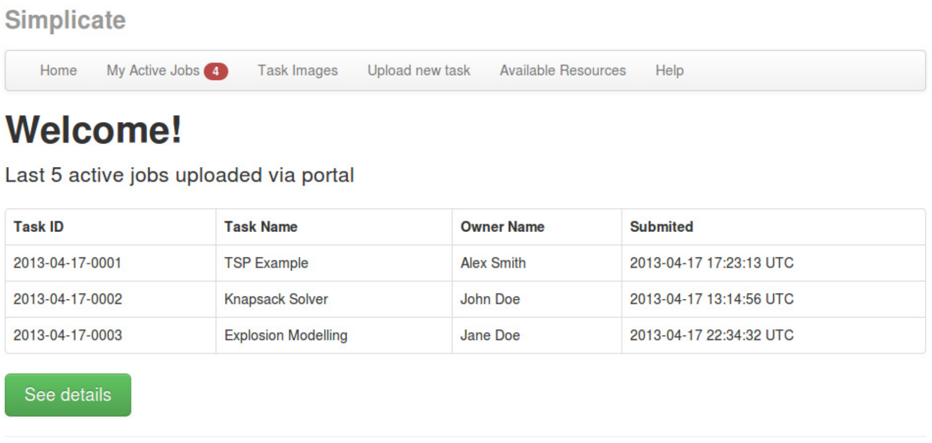
Figure 9. WMS architecture

Statistics gathering and QoS analysis module allows to predict QoS provided by resources in community grids similar to commercial grids that provide SLA (service level agreement). It is based on wrapping actual tasks into special scripts that are collecting information on each step of task execution.

Automatic priority assignment module allows to correct QoS information provided by resources if there is any by applying set of promotions and penalties based on the information received from the previously described module. It also supports user-defined priority settings.

The scheduler module is the central part of WMS which allows to schedule workflow and its subtasks efficiently and to allocate it using available resources. Based on workflow type that is analyzed by the task structure analyzer the scheduling module chooses the corresponding algorithm automatically.

In figures 10-12 screenshots of some WMS functions realization are presented. Workflow editor is a GUI tool which allows the user to define workflows by working with visual components.

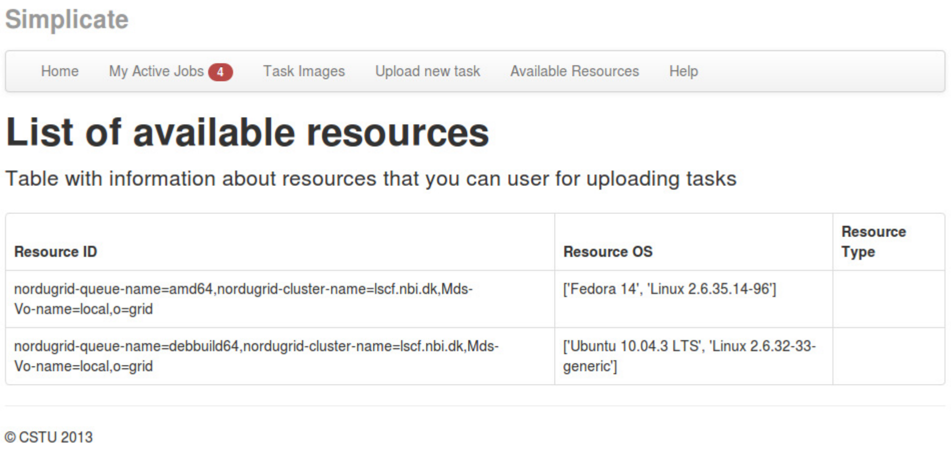


Welcome!

Last 5 active jobs uploaded via portal

Task ID	Task Name	Owner Name	Submited
2013-04-17-0001	TSP Example	Alex Smith	2013-04-17 17:23:13 UTC
2013-04-17-0002	Knapsack Solver	John Doe	2013-04-17 13:14:56 UTC
2013-04-17-0003	Explosion Modelling	Jane Doe	2013-04-17 22:34:32 UTC

Figure 10. Uploaded tasks review



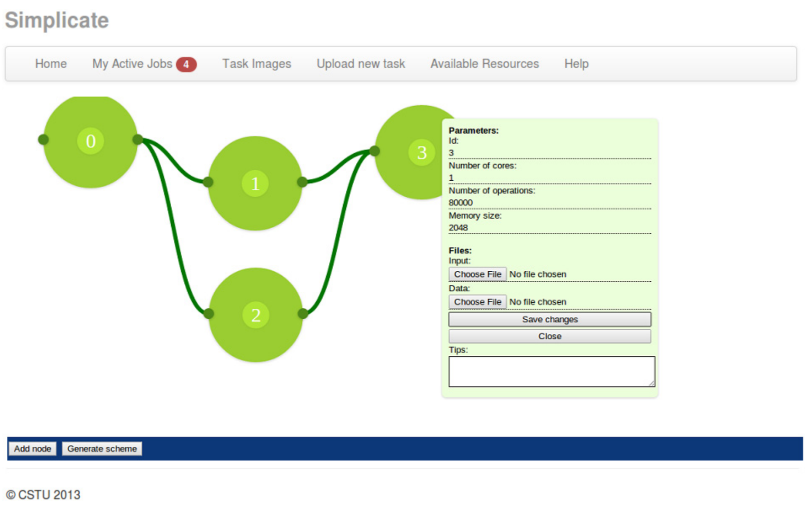
List of available resources

Table with information about resources that you can user for uploading tasks

Resource ID	Resource OS	Resource Type
nordugrid-queue-name=amd64,nordugrid-cluster-name=lscf.nbi.dk,Mds-Vo-name=local,o=grid	['Fedora 14', 'Linux 2.6.35.14-96']	
nordugrid-queue-name=debbuild64,nordugrid-cluster-name=lscf.nbi.dk,Mds-Vo-name=local,o=grid	['Ubuntu 10.04.3 LTS', 'Linux 2.6.32-33-generic']	

© CSTU 2013

Figure 11. Available resources browsing



Parameters:
Id:
3
Number of cores:
1
Number of operations:
80000
Memory size:
2048

Files:
Input:
Choose File No file chosen
Data:
Choose File No file chosen

© CSTU 2013

Figure 12. Workflow editor

Conclusion

Workflow management systems analysis shows that many of the systems (Taverna, WS-Pgrade portal, etc.) provide powerful tools for the design and implementation of the grid environment tasks such as workflow, but does not provide built-in or integrated mechanisms for effective planning which take into account the desired level of QoS.

It is worth noting that there are some task specification limitations in the systems, for example, the support for periodic synchronization between the parallel blocks is not provided.

The grid workflow scheduling algorithms research shows that choosing the scheduler algorithm based on the workflow type enables high efficiency.

In this regard, the following requirements to the developed workflow design and management system have been formulated:

1. EMI standards compliance;
2. Control point mechanism;
3. Choosing scheduler algorithm based on workflow type;
4. The ability to choose concrete resources for unit execution;
5. The possibility to choose the Runtime Environment for the task and its subtasks;
6. Dynamically redistributing units of tasks during execution;
7. Support for periodic exchange of data between the partially parallel blocks;
8. Define and consider service level requirements (taking into account QoS);
9. The ability to duplicate blocks execution in the case of high QoS requirements.

Currently, the scheduling module is implemented in C + + on the basis of the proposed QoS-based algorithm.

Further studies are planned on the classification of workflow models and the identification of the optimal scheduling algorithm according to the type of workflow model. A model of workflow execution in the grid environment based on GridSim toolkit is being developed by the authors. In the near future it is expected to receive the results of algorithms effectiveness evaluation for different types of workflows.

Acknowledgments

The authors of the paper express their gratitude to Prof. Voloshyn for the invitation to participate in the conference and discussion of the results.

Bibliography

[Appleton, 2010] O. Appleton, D. Cameron, J. Cernak, P. Dóbé, M. Ellert, T. Frågåt, M. Grønager, D. Johansson, J. Jönemo and others. The next-generation ARC middleware. Ann. Telecommun., 2010, 65:771–776, DOI 10.1007/s12243-010-0210-2.

[Cormen, 2009] Thomas H. Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. Introduction to algorithms, third ed., 2009. The MIT Press, Cambridge, Massachusetts London, England

[Curcin, 2008] V. Curcin, M. Ghanem. Scientific workflow systems - can one size fit all? Proceedings of the 2008 IEEE, CIBEC'08.

[Farkas, 2010] Z. Farkas, P. Kacsuk. P-GRADE Portal: a generic workflow system to support user communities, 2010. [Online]. Available: http://portal.p-grade.hu/download/pgportal.pdf

[Feo, 1995] T. A. Feo and M. G. C. Resende, “Greedy Randomized Adaptive Search Procedures”, Journal of Global Optimization, 6:109-133, 1995.

[Forti, 2006] Alberto Forti. DAG Scheduling for grid computing systems. Ph.D. Thesis, University of Udine – Italy, Department of Mathematics and Computer Science, 2005-2006.

[GFD-R.136, 2008] Job Submission Description Language (JSDL) Specification, Version 1.0, 2008, GFD-R.136.

[Hull, 2006] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn. Taverna: A tool for building and running workflows of services. In: Nucleic Acids Research, vol. 34, pp. W729–W732, 2006, web Server Issue.

[Jacob, 2009] Joseph C. Jacob, Daniel S. Katz, G. Bruce Berriman, John Good, Anastasia C. Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas A. Prince, Roy Williams. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. In: Int. J. Computational Science and Engineering, pp.73-87, 2009.

[Korkhov, 2010] V. V. Korkhov, D. A. Vasyunin, A. S .Z. Belloum, S. N. Andrianov, A. V. Bogdanov. Virtual Laboratory and scientific workflow management on the grid for nuclear physics applications. In: Distributed computing and grid-technologies in science and education, Proceedings of the 4th International Conference, Dubna, pp. 153–158, 2010.

[Ludäscher, 2006] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, “Scientific workflow management and the kepler system: Research articles,” Concurr. Comput. : Pract. Exper., vol. 18, no. 10, pp. 1039–1065, 2006.

[Nordugrid-Manual-4, 2013] Extended Resource Specification Language, Reference Manual for ARC versions 0.8 and above, Nordugrid-Manual-4, 2013.

[Petrenko, 2011] A.I. Petrenko, B.V.Bulah. Workflow-systems applying for the needs of modern science and engineering. In: Naukovi visti NTUU “KPI”, №5, pp.40-51, 2011.

[Shamardin, 2010] L. Shamardin, A. Demichev, A. Kryukov, V. Ilyin. GRIDNNN job execution service: a restful grid service. In: Distributed computing and grid-technologies in science and education, Proceedings of the 4th International Conference, Dubna, pp. 215–219, 2010.

[Sukhoroslov, 2010] O. V. Sukhoroslov. On development of grid-enabled applications and service-oriented scientific environments. In: Distributed computing and grid-technologies in science and education, Proceedings of the 4th International Conference, Dubna, pp. 236–240, 2010.

[Sulistio, 2004] A. Sulistio and R. Buyya, “A Grid Simulation Infrastructure Supporting Advance Reservation”, In 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), ACTA Press, Anaheim, California, November 9-11, 2004, MIT Cambridge, Boston, USA.

[Taylor, 2005] I. Taylor, M. Shields, I. Wang, and A. Harrison. Visual Grid Workflow in Triana. In: Journal of Grid Computing, vol. 3, no. 3-4, pp. 153–169, September 2005.

[Topcuoglu, 2002] H. Topcuoglu, S. Hariri, and M. Y. Wu. “Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, IEEE Transactions on Parallel and Distributed Systems, 13(3): 260-274, March 2002.

[Tsiakkouri, 2005] E. Tsiakkouri et al., “Scheduling Workflows with Budget Constraints”, In the CoreGRID Workshop on Integrated research in Grid Computing, S. Gorlatch and M. Danelutto (Eds.), Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, Nov. 28-30, 2005, pages 347-357.

[Voevodin, 2004] Voevodin V.V., Voevodin Vl.C. Parallel Computing. – St. Petersburg: BHV-Petersburg, 2004. – 608 c.

[Voloshinov, 2010] V. V. Voloshinov, S. A. Smirnov. Error-free inversion of ill-conditioned matrices in distributed computing system of restful-services of computer algebra. In: Distributed computing and grid-technologies in science and education, Proceedings of the 4th International Conference, Dubna, pp. 257–263, 2010.

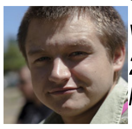
[WMS-JDL, 2011] Job description language attributes specification for the gLite Workload Management System, 2011, WMS-JDL.doc.

[Zomaya, 1999] A. Y. Zomaya, C. Ward, and B. Macey, “Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues”, IEEE Transactions on Parallel and Distributed Systems, 10(8):795-812, August 1999.

Authors' Information

Volodymyr Kazymyr *– Dr. Sc. Prof. Chernihiv State Technological University, Shevchenko street, 95, Chernihiv-27, Ukraine, 14027; e-mail: vvkazymyr@gmail.com*
Major Fields of Scientific Research: Object-oriented programming, Model-based control, Simulation

Olga Prila *– Postgraduate, the assistant lecturer, Chernihiv State Technological University, Shevchenko street, 95, Chernihiv-27, Ukraine, 14027; e-mail: olga.prila1986@gmail.com*
Major Fields of Scientific Research: Distributed computing, Simulation, Software engineering, Enterprise systems development



Volodymyr Rudyi *– MSc student, Chernihiv State Technological University, Shevchenko street, 95, Chernihiv-27, Ukraine, 14027; e-mail: vladimir.rudoy@gmail.com*
Major Fields of Scientific Research: HPC and distributed systems