Resource Scheduling Algorithm in Distributed Problem-Oriented Environments

Anastasia Shamakina and Leonid Sokolinsky

South Ural State University,
76, Lenin av., 454080 Chelyabinsk, Russian Federation
E-mail: {shamakinaav, Leonid.Sokolinsky}@susu.ru

Nowadays a large number of scheduling algorithms for the use in distributed computing environments. Only a small part of these algorithms takes into account the problem-oriented specificity. We have created a mathematical model of a computing application, which represents a workflow. The application is modelled as a weighted labelled directed acyclic graph of a job. Each task (that is a graph vertex) is labelled by a pair of natural numbers, the first one of which sets the executing time of the task on a single processor core, and the second one sets the maximum number of cores on which the task demonstrates a nearly-linear speedup. The edge weights define the amount of data, transferred between the tasks. A new Problem-Oriented Scheduling (POS) algorithm for distributed problem-oriented environments is proposed. The POS algorithm is scheduling algorithm enables to schedule tasks to run on several processor cores with task scalability limitations. The POS algorithm is designed to create a system of intelligent preparation and scheduling a workflow in order to increase the efficiency of the supercomputer complexes with cluster architecture.

1 Introduction

Today, there is a large number of scheduling algorithms [1-7] which are targeted for the use in distributed computing environments. Only a small part of these algorithms takes into account the problem-oriented specificity of workflows in complex applications. This specificity is expressed in the way of setting the execution time of a task and the amount of transmitted data.

One of the most significant achievements in this area is a Dominant Sequence Clustering (DSC) algorithm [5] invented by Yang T. and Gerasoulis A. The DSC scheduling algorithm introduces a job as a directed acyclic graph whose nodes are tasks and whose edges represent a data flow. While job scheduling, the DSC algorithm takes into account the execution time of tasks and the amount of transmitted data. A significant limitation of this algorithm applied in modern cluster computing systems is that each task can be executed only on one processor core.

We have developed a new Problem-Oriented Scheduling (POS) resource scheduling algorithm for distributed computing environments, which, unlike the DSC algorithm, allows to schedule a task to run on several processor cores with the limitations on scalability of this task.

2 Mathematical Job Model

Before a description of the POS scheduling algorithm we present a mathematical model of a job for which we need the following basic definitions.

2.1 Basic Definitions

A directed graph is called quadruple $G = \langle V, E, init, fin \rangle$, where V is a vertices set; E is an edges set; $init : E \to V$ is a function which determines *an initial vertex* of an edge; $fin : E \to V$ is a function which determines *a final vertex* of an edge.

The vertices $\nu, \nu^{'} \in V$ are called *adjacent*, if

$$\exists e \in ((\nu = init(e) \& \nu^{'} = fin(e)) \bigvee (\nu = fin(e) \& \nu^{'} = init(e))), \tag{1}$$

in other words, there is an edge e connecting these vertices. If $\nu = init(e) \& \nu^{'} = fin(e)$, we denote it as follows: $(\nu, \nu^{'}) = e \in E$ and we say that the vertices $\nu, \nu^{'}$ are *incident* to the edge e.

Let us take the vertices $\nu, \nu^{'} \in V$ and the number $n \geq 1$. An ordered sequence of edges $(e_1, e_2, \ldots, e_n) \in E^n$ is called *a path of a length n* from the vertex ν to the vertex $\nu^{'}$, if $\nu = init(e_1)$, $\nu^{'} = fin(e_n)$ and $fin\,(e_i) = init(e_{i+1})$ for all $i \in \{1, \ldots, n-1\}$. If $(e_1, e_2, \ldots, e_n) \in E^n$ is a path from the vertex ν to the vertex $\nu^{'}$, then *a return path* from the vertex $\nu^{'}$ to the vertex ν is called an ordered sequence of edges $(e_n, e_{n-1}, \ldots, e_1) \in E^n$. The path $(e_1, e_2, \ldots, e_n)$ is called *simple one*, if all vertices $init(e_1), init(e_2), \ldots, init(e_n)$ different from each other and if all vertices $fin(e_1), fin(e_2), \ldots, fin(e_n)$ also distinct. *A circle* is called a simple path from some vertex to itself. A directed graph is called *acyclic one*, if it does not contain cycles [8].

The vertices $\nu, \nu^{'} \in V$ are called *independent ones*, if there is no simple or return paths from the vertex ν to the vertex $\nu^{'}$. Otherwise, the vertices $\nu, \nu^{'}$ are *dependent*.

Let us take a directed graph $G = \langle V, E, init, fin \rangle$. *The weighting of the graph* G is called a function $\delta : E \to \mathbb{Z}_{\geq 0}$. *A layout of the graph* G is called a function

$$\gamma : V \to \mathbb{N}^2. \tag{2}$$

2.2 Computing Environment Model

Now, we are ready to give a mathematical definition of a job graph in a distributed computing environment.

A job graph is called a marked-up weighted directed acyclic graph,

$$G = \langle V, E, init, fin, \delta, \gamma \rangle,$$

where V is a set of vertices which correspond to tasks, E is a set of edges which correspond to data flows. *The weighting function* $\delta\,(e)$ of the edge e determines the amount of transmitted data from a task associated with the vertex $init(e)$ to a task associated with the vertex $fin(e)$. *A layout*

$$\gamma\,(\nu) = (m_\nu, t_\nu) \tag{3}$$

determines the maximum number of processor cores m_ν on which the task ν has *a nearly-linear speedup* of the execution time t_ν of task ν on a single core. In this model, we assume that *the computational cost* $\chi\,(\nu, j_\nu)$ of the task ν on $j_{\nu-th}$ processor cores is determined by the following formula:

$$\chi(\nu, j_\nu) = \begin{cases} t_\nu / j_\nu, & \text{if } 1 \leq j_\nu \leq m_\nu; \\ t_\nu / m_\nu, & \text{if } m_\nu < j_\nu. \end{cases} \tag{4}$$

In other words, the execution time decreases directly proportional to the increasing the number of processor cores in the range from 1 to m_ν. We will not get the acceleration by increasing the number of processor cores in the range from m_ν to $+\infty$.

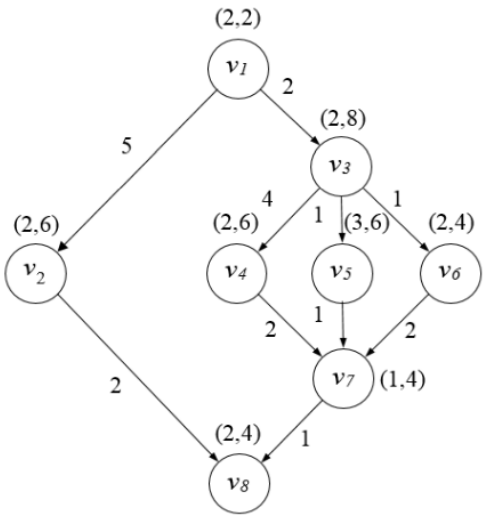

Figure 1. An example of a job graph.

Fig. 1 shows an example of job graph consisting of 8 vertices. A label is a pair of numbers (m_ν, t_ν) specified for each of the vertices. Each edge has a weight. The weight $\delta(e)$ determines the amount of data transmitted to the edge e.

The compute node P is an ordered set of processor cores $\{c_0, \ldots, c_{d-1}\}$.

The computer system $\mathfrak{P}$ is an ordered set of compute nodes $\{P_0, \ldots, P_{k-1}\}$. In reality, the computer system may be a distributed computing system, combining several computing clusters, each of which is a separate node on this system.

The clustering is called single-valued transformation $\omega : V \to \mathfrak{P}$ of vertices set V of the job graph G on a set of computational nodes $\mathfrak{P}$.

The computer system $\mathfrak{P} = \{P_0, \ldots, P_{k-1}\}$, consisting of k computational nodes, is given. *The cluster* W_i is a set of all vertices that are displayed on the computational node $P_i \in \mathfrak{P}$:

$$W_i = \{\nu \in V | \omega(\nu) = P_i \in \mathfrak{P}\}. \tag{5}$$

We have the following properties:

$$W_i \bigcap W_j = \emptyset \text{ for } i \neq j; \tag{6}$$

$$V = \bigcup_{i=0}^{k-1} W_i. \tag{7}$$

Let us consider the job graph $G = \langle V, E, init, fin, \delta, \gamma \rangle$ with the clustering function ω. We will call such job graph as *clustered one* and denote as $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$.

In this model, we assume that the transfer time of any amount of data between nodes belonging to the same cluster is close to zero, and the transmission of data between nodes belonging to different clusters is proportional to the volume of transmitted data with coefficient 1. Based on this, we can define *the communication cost function* $\sigma : E \to \mathbb{Z}_{\geq 0}$, which computes the communication cost (time) of transmitted data along the edge $e \in E$ as follows:

$$\sigma(e) = \begin{cases} 0, & \text{if } \omega(init(e)) = \omega(fin(e)); \\ \delta(e), & \text{if } \omega(init(e)) \neq \omega(fin(e)). \end{cases} \tag{8}$$

The clustered graph $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$ is given. *A schedule* is called single-valued transformation, $\xi : V \to \mathbb{Z}_{\geq 0} \times \mathbb{N}$, which maps the casual vertex $v \in V$ on a pair of numbers,

$$\xi(v) = (\tau_v, j_v), \tag{9}$$

where τ_v determines the launch time of the task v and j_v is a number of processor cores allocated to task v. We denote by s_v the stop time of the task v. Then

$$s_v = \tau + \chi(v, j_v), \tag{10}$$

where χ is the computational cost function defined by the formula 4. A schedule is called *valid one*, if it satisfies the following conditions:

$$\forall\, e \in E \ \left(\tau_{fin(e)} \geq \tau_{init(e)} + \chi\left(init(e), j_{init(e)}\right) + \sigma(e)\right); \tag{11}$$

$$\forall\, v \in V \left(j_v \leq m_v\right); \tag{12}$$

$$\forall\, t \in \mathbb{N} \left(\forall\, i \in [0, \ldots, k-1] \left(\sum_{v \in W_i \& \tau_v < t \leq s_v} j_v \leq |P_i| \right) \right). \tag{13}$$

The condition 11 means that for any two adjacent vertices $v_1 = init(e)$ and $v_2 = fin(e)$ the start time v_2 cannot be less than the sum of the following values: the start time of the task v_1, the execution time of the task v_1, and the communication cost of the edge e. The condition 12 means that the number of cores allocated the task v_1, does not exceed the boundaries of linear scalability, which sets the layout γ in the context of the formula 3. The condition 13 means that at any time t the total number of processor cores allocated by tasks on the node number i cannot exceed the number of cores on the same node. In what follows we consider only valid schedules, unless explicitly stated otherwise.

The clustered graph $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$ with the specified schedule ξ is called *a scheduled graph*, $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$.

A tiered-and-parallel form of a graph [9] is called a partition of vertices set V of the directed acyclic graph $G = \langle V, E, init, fin \rangle$ into renumbered subsets (tiers) L_i, where $i = 1, \ldots, r$, satisfying the following properties:

$$\left.\begin{array}{l} V = \bigcup\limits_{i=1}^{r} L_i; \\ \forall i \neq j \in \{1, \ldots, r\} (L_i \bigcap L_j = \emptyset); \\ \forall (v_1, v_2) \in E (\forall i \neq j \in \{1, \ldots, r\} (v_1 \in L_i \& v_2 \in L_j \Rightarrow i < j)). \end{array}\right\} \tag{14}$$

The last condition means that if there is an edge from the vertex v_1 to the vertex v_2, then the vertex v_2 must be placed on a tier with a large number in relation to a tier on which the vertex v_1 is placed. A number of vertices in a tier L_i is called its *width*. A number of tiers in a tiered-and-parallel form is called its *height*, and the maximum of a width of its tiers is called *a width of a tiered-and-parallel form*. A tiered-and-parallel form is called *canonical one* [10], if all input vertices (without input edges) belong to the tier 1 and the maximum path length, ending at a vertex owned tier k, equals $k-1$.

Fig. 2 presents an example of a canonical tiered-and-parallel form consisting of 5 tiers for the graph shown in Fig. 1.

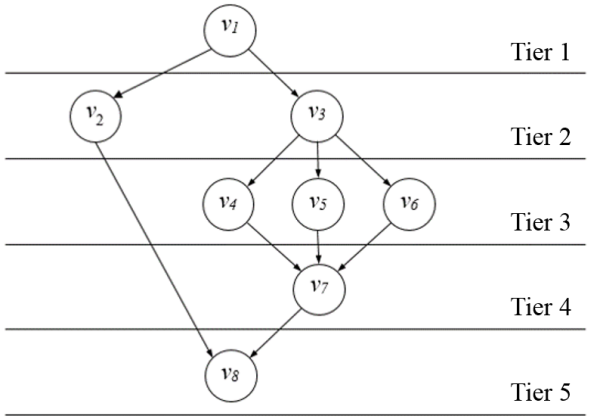

Figure 2. An example of a canonical tiered-and-parallel form.

Let us take a simple path, $y = (e_1, e_2, \ldots, e_n)$, in the scheduled graph $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$. Since the graph G is acyclic, any path in it, including the path y, will be the simple one. *The path cost* $u(y)$ has the following value:

$$u(y)=\chi\left(fin(e_n), j_{fin(e_n)}\right) + \sum_{i=1}^{n}\left(\chi\left(init(e_i), j_{init(e_i)}\right) + \max(\sigma\left(e_i\right), \tau_{fin(e_i)} - s_{init(e_i)})\right), \quad (15)$$

where χ is a function defined by the formula 4, whose value is a computational cost of a vertex; σ is a function defined by the formula 8, whose value is a communication cost of edge; j_v and τ_v are defined by the formula 9; s_v is defined by the formula 10.

A set Y of all simple paths in scheduled graph $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$ is given. The simple path $\bar{y} \in \mathrm{Y}$ is called *a critical path*, if it has a maximum value:

$$u(\bar{y})=\max_{y \in Y} u(y)\ . \quad (16)$$

3 Problem-Oriented Scheduling Algorithm

In this section, we will describe a new POS (Problem-Oriented Scheduling) algorithm for the scheduling in distributed problem-oriented environments. A distinctive feature of the POS algorithm is that it takes into account the knowledge about the specificity of a subject area. In the model described in Subsection 2.2 this specificity is expressed in the way of setting the execution time of a task and the amount of transmitted data.

To simplify a description and an understanding of the POS algorithm, we will use the three-level structure of procedures representing the POS algorithm. The procedure of the first level is *the main procedure*. Some steps in the first level will be described in the second level. We will highlight such steps in bold. A similar approach will be used in a description of the second level.

3.1 Main Procedure

Let us consider the computer system $\mathfrak{P} = \{P_0, \ldots, P_{k-1}\}$ in the form of an ordered set of computing nodes. The job graph $G = \langle V, E, init, fin, \delta, \gamma \rangle$ is given. We assume that the following conditions are fulfilled:

$$|V| \leq |\mathfrak{P}|, \tag{17}$$

$$\forall v \in V(\forall P \in \mathfrak{P}(m_v \leq |P|)), \tag{18}$$

where m_v is a boundary linear scalability defined by the layout function γ. We define for graph G a canonical tiered-and-parallel form with tiers $L_i (i = 1, \ldots, r)$. Then we enumerate vertices $V = (v_1, \ldots, v_q)$ of the graph G so that the following property is fulfilled:

$$\forall i, j \in \{1, \ldots, q\}((v_i \in L_a \& v_j \in L_b \& a < b) \Rightarrow i < j), \tag{19}$$

this has to result that vertices with large numbers are arranged on the lower tiers.

Tab. 1 presents the main procedure in the most general form.

Step 1. **Initial configuration** $G_0 = \langle V, E, init, fin, \delta, \gamma, \omega_0, \xi_0 \rangle$;
Step 2. $i := 0$;
Step 3. **Iterate over configurations: from** $G_i = \langle V, E, init, fin, \delta, \gamma, \omega_i, \xi_i \rangle$ **to** $G_{i+1} = \langle V, E, init, fin, \delta, \gamma, \omega_{i+1}, \xi_{i+1} \rangle$**.** **The considered edges will be marked as examined;**
Step 4. If there are still unconsidered edges remaining, then
Step 4.1. $i := i + 1$;
Step 4.2. go to step 3;
Step 5. **Compressing** $G_{i+1} = \langle V, E, init, fin, \delta, \gamma, \omega_{i+1}, \xi_{i+1} \rangle$**;**
Step 6. The end of the procedure.

Table 1. The main procedure.

Thus, the purpose of the main procedure is to construct a sequence of configurations. In this case, during the transition from one to another configuration at least one edge of

a graph is marked as examined. Since the number of edges is finitely, the procedure is completed at some iteration. Last built configuration G_{i+1} is selected as the resultant.

3.2 Initial Configuration

Tab. 2 presents the procedure of preparation of the initial configuration G_0.

Step 1.1. We define a function of an initial clustering as follows:
$\forall i \in \{1, \ldots, q\}(\omega_0(v_i) = P_{i-1})$.
that is, each vertex is displayed on a separate computing node,
and accordingly, each cluster includes only one vertex.

Step 1.2. We specify τ_v and j_v for an initial schedule $\xi_0(v) = (\tau_v, j_v)$.
Then recursively define start time τ_v by tiers of a tiered-and-parallel form:

$$\left.\begin{array}{l} \forall v \in L_1(\tau_v := 0); \\ \forall v \in L_{i>1}(\tau_v := \max\limits_{v'' \in L_i; v' \in L_{j<i}} (\lambda(v', v''))). \end{array}\right\}$$

Here

$$\lambda(v', v'') = \begin{cases} s_{v'}, if\ (v', v'') \notin E; \\ s_{v'} + \sigma((v', v'')), if\ (v', v'') \in E. \end{cases}$$

where $s_{v'}$ is defined by the formula 10. We determine the number of processor cores j_v allocated to the vertex v as follows: $\forall v \in V(j_v = m_v)$.

Step 1.3. $G_0 = \langle V, E, init, fin, \delta, \gamma, \omega_0, \xi_0 \rangle$.

Step 1.4. The end of the procedure.

Table 2. The procedure for preparing the initial configuration G_0.

3.3 G_{i+1} Configuration

We define *a subcritical path* as a path having the maximum cost and containing at least one unexamined edge. Tab. 3 presents the procedure for preparing the configuration G_{i+1}.

3.4 ξ_{i+1} Schedule

We introduce the following notation: $T(x)$ is the tier number, which the vertex x owns; $W_{\omega_i(x)} = \{v | v \in V, \omega_i(v) = \omega_i(x)\}$ is the cluster, which the vertex x owns (Fig. 4).

3.5 G_{i+1} Compressing

The aim of the compressing procedure is to minimize the number of involved computing nodes. This procedure applies only to clusters containing one vertex. This limitation is motivated by the fact that if there are two adjacent vertices in a cluster, then the movement one of them to another cluster may increase the total task time.

Let us give some explanation to the compressing procedure. Step 5.2 organizes a loop, building cluster set $\mathfrak{M}$, which is a partition of the tasks set V (Fig. 5).

Step 3.1. Find a subcritical path for the graph $\tilde{y}_i = (e_1, \ldots, e_n)$;
Step 3.2. Find the first unexamined edge e_j $(1 \le j \le n)$ from the beginning of path $\tilde{y}_i$ and marking it as examined one;
Step 3.3. If $i = 0$, then mark vertex $init(e_j)$ as fixed one;
Step 3.4. If vertices $init(e_j)$ and $fin(e_j)$ are fixed, then go to step 3.14;
Step 3.5. If the vertex $fin(e_j)$ is not fixed, then $v'' := fin(e_j), v' := init(e_j)$;
Step 3.6. If the vertex $init(e_j)$ is not fixed, then $v'' := init(e_j), v' := fin(e_j)$;
Step 3.7. Specify the function of clustering ω_{i+1}, which equals the function of clustering ω_i, excepting $\omega_{i+1}(v'') := \omega_i(v')$;
Step 3.8. **Prepare the schedule** ξ_{i+1};
Step 3.9. $G_{i+1} = \langle V, E, init, fin, \delta, \gamma, \omega_{i+1}, \xi_{i+1} \rangle$;
Step 3.10. Find the critical path $\bar{y}_i$ in the graph G_i;
Step 3.11. Find the critical path $\overline{y_{i+1}}$ in the graph G_{i+1};
Step 3.12. If $u(\overline{y_{i+1}}) \le u(\bar{y}_i)$, then go to step 3.16;
Step 3.13. $G_{i+1} := G_i$;
Step 3.14. If there are unexamined edges in path $\tilde{y}_i$ then go to step 3.2;
Step 3.15. If there are unexamined edges in graph G_i, then go to step 3.1;
Step 3.16. The end of the procedure.

Table 3. The procedure for preparing the configuration G_{i+1}.

Step 3.8.1. $R := W_{\omega_i(v')} \bigcap L_{T(v'')}$;
Step 3.8.2. If $R = \emptyset$ or $\sum\limits_{v \in R} j_v \le |P_{\omega_i(v')}|$, then go to step 3.8.7;
Step 3.8.3. For $h = q, \ldots, T(fin(e_j)) + 1$ perform $L_{h+1} := L_h$;
Step 3.8.4. $L_{T(v'')+1} := \{v''\}$;
Step 3.8.5. $L_{T(v'')} := L_{T(v'')} \setminus \{v''\}$;
Step 3.8.6. $q := q + 1$;
Step 3.8.7. Specify new schedule ξ_{i+1} by calculating the start time τ_v of all vertices $v \in V$ using the formula (21);
Step 3.8.8. Mark vertex v'' as fixed one;
Step 3.8.9. The end of the procedure.

Table 4. The procedure for preparing the schedule ξ_{i+1}.

Step 5.1. $\mathfrak{M} := \emptyset$;
Step 5.2. For each vertex $v' \in V$ perform a loop
Step 5.2.1. $W = \{v | v \in V, \omega_{i+1}(v) = \omega_{i+1}(v')\}$;
Step 5.2.2. If $W \in \mathfrak{M}$, then go to next iteration of the loop;
Step 5.2.3. $\mathfrak{M} := \mathfrak{M} \bigcup \{W\}$;
Step 5.3. The end of the loop (step 5.2);

Table 5. Building cluster set $\mathfrak{M}$.

Tab. 6 presents the compressing procedure of graph G_{i+1}. Step 5.4 computes set $\mathfrak{E}$ of *single clusters* (clusters containing only one vertex) and set $\mathfrak{V}$ of *multiclusters* (clusters comprising two or more vertices). Step 5.5 loops through all single clusters. For each single cluster we find a tier of a tiered-and-parallel form, to which this single cluster belongs. Within this tier we try to combine the single cluster with some multicluster. It is possible, if a multicluster does not use all processor cores and the number of free cores is sufficient to perform the single cluster.

Step 5.4. $\mathfrak{E} := \{W \in \mathfrak{M} || W_a| = 1\}$; $\mathfrak{V} := \{W \in \mathfrak{M} || W_a| > 1\}$;
Step 5.5. For each $W' \in \mathfrak{E}$ perform a loop
Step 5.5.1. For $l = 1, \ldots, r$ perform a loop
Step 5.5.1.1. If $W' \bigcap L_l = \emptyset$, then go to next iteration of the loop;
Step 5.5.1.2. For each $W'' \in \mathfrak{V}$ perform a loop
Step 5.5.1.2.1. If $W'' \bigcap L_l = \emptyset$, then go to next iteration of the loop;
Step 5.5.1.2.2. Get $v' \in W'$ and $v'' \in W''$;
Step 5.5.1.2.3. If $j_{v'} + \sum\limits_{v \in W'' \bigcap L_l} j_v > \omega_{i+1}(v'')$, then transfer to next iteration of the loop;
Step 5.5.1.2.4. $i := i + 1$;
Step 5.5.1.2.5. Specify the function of clustering ω_{i+1}, which equals the clustering function ω_i, exepting $\omega_{i+1}(v') := \omega_i(v'')$;
Step 5.5.1.2.6. $W'' := W'' \bigcup W'$;
Step 5.5.1.2.7. Go to step 5.5.3;
Step 5.5.1.3. The end of the loop (step 5.5.1.2);
Step 5.5.2. The end of the loop (step 5.5.1);
Step 5.5.3. Go to next iteration of the loop;
Step 5.6. The end of the loop (step 5.5);
Step 5.7. The end of the procedure.

Table 6. The procedure of compressing G_{i+1}.

4 Integration with UNICORE

The POS algorithm has been implemented as a Grid service of the Distributed Virtual Test Bed (DiVTB) system [12, 11] on the basis of the UNICORE platform [13, 14] using Java programming language.

The DiVTB (Distributed Virtual Test Bed) is a software system that ensures the development and operation of distributed virtual test beds. DiVTB provides for a task-oriented approach of solving specific classes of problems in engineering design through resources supplied by grid computing environments.

The DiVTB system includes the following components (Fig. 3).

A DiVTB Developer which is a web-application that provides for the development for DiVTBs for a grid environment. An application programmer is given an opportunity to visually design the workflow for simulation, to create a file template of a task statement, to generate a set of parameters of a simulated task. The system also enables a function of the export DiVTB to a DiVTB Server where they can be launched for task calculation.

A DiVTB Portal which is a web-application for an engineer, which provides for start-up and reception of simulation results of virtual experiments. The DiVTB Portal ensures of authentication, user account management and a user interface for virtual experiments by means of DiVTB. The DiVTB Portal incorporates a built-in web-form generator which automatically generates a user interface on the basis of a relevant DiVTB's parameter description.

A DiVTB Viewer which is an auxiliary service for interactive visualisation of distributed virtual test beds. The DiVTB Viewer is designed to check for a correct display of generated problem-oriented shells of a DiVTB.

A DiVTB Server which is an environment for storing of DiVTBs and managing virtual experiments. The DiVTB Server ensures the execution of a DiVTB workflow, including the formatting of a file of task statement for each individual computing service in a grid environment on the basis of experimental parameters defined by an engineer, as well as the obtaining of simulation results and transfer of results to an engineer.

A DiVTB Broker which implements automated registration, search for, and allocation of grid resources to perform the actions of engineering simulation. The DiVTB Broker maintains a permanent record of grid resources which are available in the computing environment, and provides sets of computing resources at the demand of the DiVTB Server for carrying out computational experiments [6]. The DiVTB Broker component obtains information about existing applications of a grid computing environment and resources from the Registry and UNICORE/X Site components of the UNICORE.

In the Fig. 3 the components of the UNICORE platform that intend to cooperate with the DiVTB system are marked with red colour.

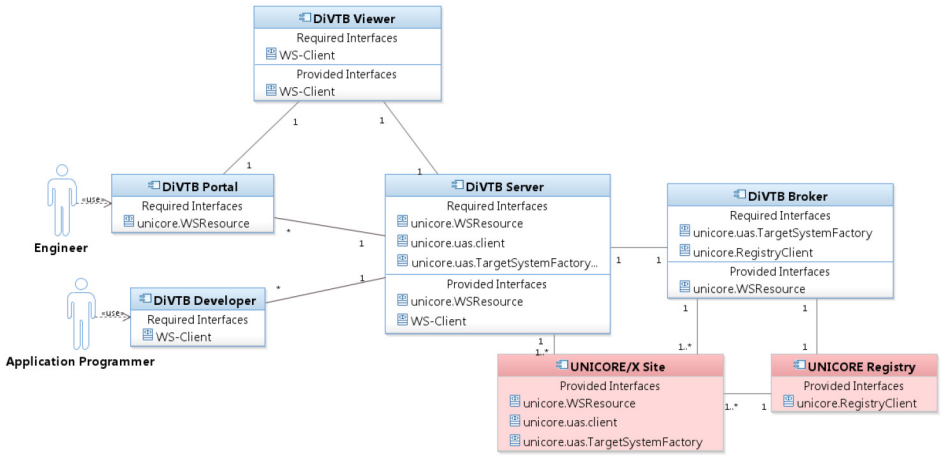

Figure 3. The architecture of the DiVTB system.

Architecture of the DiVTB Broker shown in the Fig. 4. The DiVTB Broker consists of the following components.

- The Master accepts requests from the DiVTB Server and creates an instance of a

WSResource that is called a Brokered Workflow.

- Each Brokered Workflow processes one request. It generates a list of required resources to perform a workflow and make the reservation of resources.
- The Resource Manager works with the Resource Database which contains the information about target systems and reserved resources.
- The Statistics Manager supports the Statistics Database which contains the information about tasks statistics (the execution time, the amount of transmitted data, the types of graphs).
- The Collector works independently from the DiVTB Broker and carries out the collection of the information for the Resource Database.

The DiVTB Broker creates a schedule for an initial graph using by the Resource Database and the Statistics Database.

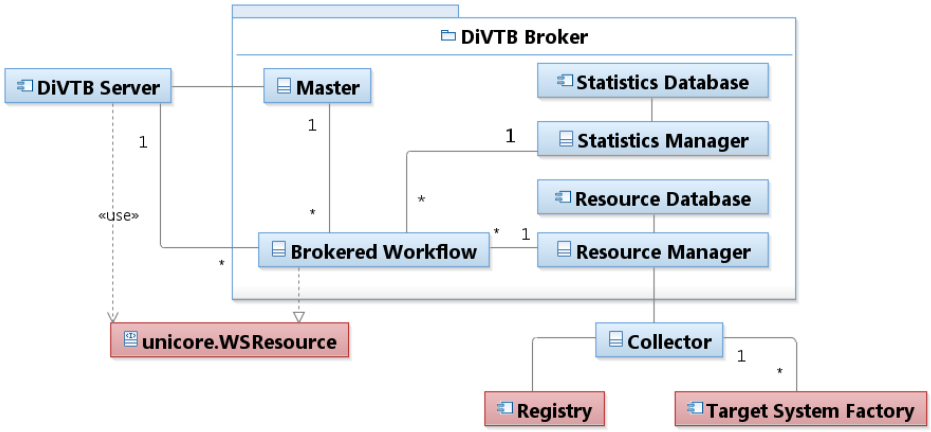

Figure 4. The architecture of the DiVTB Broker.

5 Conclusions

A developed mathematical model allows to describe a workflow as a job graph and to involve the use of vertex clustering; the ability to specify an execution time of each task and its upper limit of linear scalability; the ability to set the number of processor cores for each compute nodes. This mathematical model is a universal one, because it provides an opportunity to describe already existing and new algorithms for clustering, as well as new algorithms for scheduling of resources in distributed computing environments.

The POS (Problem-Oriented Scheduling) algorithm for distributed cluster computing environments allows us to schedule one task to run on multiple cores with the limitations

on its scalability. This scheduling algorithm is designed to create a system of intelligent building and scheduling a workflow in order to increase the efficiency of the supercomputer complexes with cluster architectures.

Acknowledgements

The present work was supported by Russian Foundation for Basic Research (RFBR), Research Project No. 14-07-31159-mol-a "My first grant".

References

1. S.J. Kim, *A general approach to multiprocessor scheduling*, Report TR-88-04. Department of Computer Science, University of Texas at Austin, 1988.
2. S.J. Kim, J.C. Browne, *A general approach to mapping of parallel computation upon multiprocessor architectures*, Proceedings of the International Conference on Parallel Processing, Volume 3, pp. 1-8, 1988.
3. V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, The MIT Press, Cambridge, MA, P. 215, 1989.
4. A. Gerasoulis, T. Yang, *A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors*, J. Parallel and Distributed Computing **16(4)**, 276–291, 1992.
5. T. Yang, A. Gerasoulis, *DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors*, Proceedings of the IEEE Transactions on Parallel and Distributed Systems, Volume 5, No. 9. pp. 951-967, 1994.
6. A. Shamakina, *Brokering Service for Supporting Problem-Oriented Grid Environments*, Proceedings of the 2012 UNICORE Summit, IAS Series, Volume 15, pp. 67-75, 2012.
7. A.V. Shamakina, *Brokering Service for Supporting Problem-Oriented Grid Environments* J. Bulletin of South Ural State University. Series: Computational Mathematics and Informatics **46(305)**, 88–98, 2012 (in Russ.).
8. D.E. Knuth, *The Art of Computer Programming*, Volume 1: Fundamental Algorithms, 3rd Edition, P. 652, 1997.
9. V.V. Voevodin, *Mathematical models and methods in parallel processes*, Science, P. 296, 1986.
10. V.V. Voevodin, Vl.V. Voevodin. *Parallel computing*, St. Petersburg, BHV- Petersburg, P. 608, 2002.
11. G. Radchenko and E. Khudyakova, *Distributed Virtual Test Bed: an Approach to Integration of CAE Systems in UNICORE Grid Environment* Proceedings of the 36th International Convention MIPRO 2013, pp. 183-188, 2013.
12. G. Radchenko and E. Khudyakova, *A Service-Oriented Approach of Integration of Computer-Aided Engineering Systems in Distributed Computing Environments* Proceedings of the 2012 UNICORE Summit, IAS Series, Volume 15, pp. 57-66, 2012.
13. The UNICORE platform, see `www.unicore.eu/`
14. K. Benedyczak, P. Bała, S. van den Berghe, R. Menday, and B. Schuller, *Key aspects of the UNICORE 6 security model*, J. Future Generation Comp. Syst. **27(2)**, 195–201, 2011.