Advances in Engineering Research, volume 181

International Conference on Electronical, Mechanical and Materials Engineering (ICE2ME 2019)

A Modified Method of Multiagent Resource Dispatching in a Heterogeneous Cloud Environment

I.A. Kalyaev[1], A.I. Kalyaev[2] and I.S. Korovin[2,*]

[1]Southern Scientific Center of the Russian Academy of Sciences, Taganrog, Russia

[2]Southern Federal University, Russia

[*]Corresponding author

***Abstract*—At present the level of development of telecommunication technologies gives new opportunities for organization of distributed calculations, based on a set of space-distributed computational resources, with the help of a service-oriented infrastructure, which provides "calculations on demand". Owing to these opportunities a new paradigm of cloud calculations has appeared: large-scale distributed calculations based on a pool of abstract, virtualized, dynamically redistributable computational resources, granted to external users via Internet on their demands. Nowadays cloud computing environments (CCE) get wider and wider application for solution of large-scale scientific and technical problems in various domains, such as high-energy physics, chemistry and biology; cosmology and astrophysics; ecological and technical safety; industry, pharmacy; material science; oil and gas production; medicine, etc. [2]–[7]. The specific feature of such large-scale scientific problems is their complex inner structure, which, as a rule, consists of several data coupled subtasks. Besides, it happens rather often that efficiency of such subtasks considerably depends on the type of computational resource, which is used for their implementation. One of the main advantages of the concept of cloud calculations is applicability of different-type (heterogeneous) computational resources within one CCE. In general case such computational resources can have different real performance for various tasks. This circumstance, on the one hand, allows to gain in efficiency (time decreasing) of implementation of large-scale user tasks, owing to use of such CCE computational resources, which provide the highest real performance for these tasks (or for their subtasks). However, on the other hand, it causes the problem of optimal distribution (dispatching) of solving tasks (or their subtasks) to different-type computational resources of the CCE. The problem becomes much more complex in the case, when the CCE has to provide solution of a certain set (flow) of miscellaneous user tasks coming in arbitrary time, but not the only task. It requires development of new methods and algorithms of adaptive resource dispatching for heterogeneous CCEs for processing of a flow of large-scale tasks. In this paper we suggest a method of multi-agent resource dispatching in a heterogeneous CCE, which provides automatic distribution of the CCE resources during implementation of the user tasks flow. Such multi-agent organization of the dispatcher provides quasi-optimal adaptive distribution of the CCE resources and takes into account their real performance and bandwidth capacity of their channels; high productive loading of the CCE resources; capability of unlimited scaling of the CCE heterogeneous computational resources; improved fault-tolerance of the CCE, because it contains no nodes, that can fail and cause catastrophic consequences for the whole system. However, we assume that Customers set the time when they require their tasks to be solved. This requirement can lead to the situation, where it is impossible to solve some tasks because Customers have set unacceptable time for their solution. As a result, it will cause decreasing of efficiency of the whole CCE. That is why in the paper we consider a more general statement of the problem of CCE resource dispatching, when the required solution time for user tasks is not set, and the CCE dispatcher has to minimize the time of their solution with the help of available computational resources of the CCE.**

Keywords—component; formatting; style; styling; insert

I. FORMAL TASK STATEMENT

Let us assume that the CCE receives a certain set (flow) of different large-scale tasks $\mathbf{Z} =< Z_1, Z_2, ..., Z_M >$ from different Customers in different time. Here a large-scale task is a task, which consists of a certain set of information-dependent (coupled) subtasks, each with a considerable computational complexity. Formally, each large-scale task $Z_l \in \mathbf{Z}$ can be represented in the form of an acyclic graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ (see Figure I). Its vertices $q_j \in \mathbf{Q}_l$ correspond to some subtasks O_j, which belong to the subtask set $\mathbf{O} =< O_1, O_2, ..., O_c >$. Its edges $x(q_j, q_{j+1}) \in \mathbf{X}_l$ determine information connections among the subtasks (i.e., if two vertices q_j and q_{j+1} are connected by the edge $x(q_j, q_{j+1})$, then it means that the results of the subtask O_j, which corresponds to the vertex q_j, is initial data for the subtask O_{j+1}, which corresponds to the vertex q_{j+1}. Let us consider, that each vertex $q_j \in \mathbf{Q}_l$ has a type of a solving subtask $O_j \in \mathbf{O}$ and its computational complexity v_j, which is evaluated by the number of elementary computational operations performed for solution of the subtask. Besides, each edge $x(q_j, q_{j+1}) \in \mathbf{X}_l$ has data amount $d_{j,j+1}$, transferred from the subtask O_j, assigned to the vertex q_j, to the subtask O_{j+1}, assigned to the vertex q_{j+1} (see Figure I).

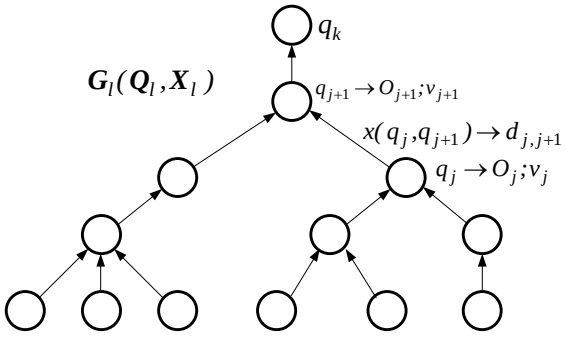

FIGURE I. THE GRAPH $\mathbf{G}_l^2(\mathbf{Q}_l^2, \mathbf{X}_l^2)$ OF THE TASK Z_1

Let the CCE contain several heterogeneous computational resources $\mathbf{R} =< R_1, R_2, ...R_N >$. Here a computational resource is some computational device (a personal computer, a server, a multiprocessor computer system, etc.), connected to the cloud infrastructure. Let us consider, that each resource $R_i \in \mathbf{R}$ can solve a certain set (subset) of subtasks $\mathbf{O}_i =< O_1, O_2, ...O_L > \subseteq \mathbf{O}$, $(i = 1, 2, ..., N)$. The real performance of a resource R_i during implementation of a subtask $O_j \in \mathbf{O}_i$ $(j = 1, 2, ..., L)$ is $S_i(O_j)$ (operations per second). In the general case, real performance of different CCE resources during implementation of identical subtasks O_j can also be different, i.e. $S_p(O_j) \neq S_c(O_j)$ $(p = 1, 2, ..., N; c = 1, 2, ..., p-1, p+1, ..., N)$. Besides, let us assume that we know the bandwidth capacity Y_p (byte per second) of the channel, which connects the resource R_i $(i = 1, 2, ..., N)$ with the cloud infrastructure.

The aim of the CCE dispatcher is to minimize the solution time of all tasks of the set $\mathbf{Z}$ by rational distribution of their subtasks to the computational resources $\mathbf{R} =< R_1, R_2, ..., R_N >$, taking into account their real performance for any subtask and the bandwidth capacity of the channels which connect the resources and the cloud infrastructure.

It is rather difficult to solve the problem of CCE resource distribution to the tasks of the flow $\mathbf{Z}$, using traditional methods such as methods of linear or dynamical programming [10], [11]. First of all, the number of computational resources of the CCE and the number of different tasks can be rather large, and it leads to exponential increase of search variants. Secondly, since the tasks are coming in a-priori unknown time, it is impossible to form a schedule of their implementation in advance, and it is necessary to distribute dynamically the subtasks of the different tasks of the set $\mathbf{Z}$ to the resources $\mathbf{R} =< R_1, R_2, ...R_N >$ taking into account their functional capabilities and their performance for the corresponding task. That is why in this case it is reasonable to use the multi-agent approach to the problem of CCE resource dispatching which simulates principles of social behaviour of human communities during execution of some complex tasks [8], [12]. (See Figure II)

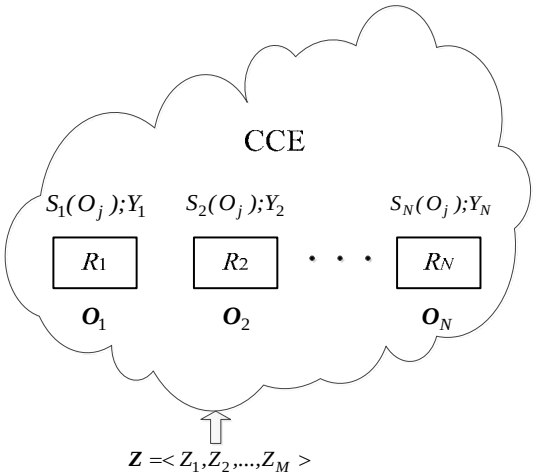

FIGURE II. CLOUD COMPUTING ENVIRONMENT

II. THE MULTIAGENT CCE DISPATCHER FUNCTIONING PRINCIPLES

Before development of a new algorithm of multi-agent CCE dispatching, let us widen the notion "thread", introduced in [8], [9]. As normal, a thread is some sequence of vertices $\mathbf{H}_f =< q_1^f, q_2^f, ..., q_k^f >$ of the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ of the task $Z_l \in \mathbf{Z}$, where the vertices q_j^f and q_{j+1}^f $(j = 1, 2, ..., k-1)$ are connected by the edge $x(q_j^f, q_{j+1}^f)$ (see Figure III).

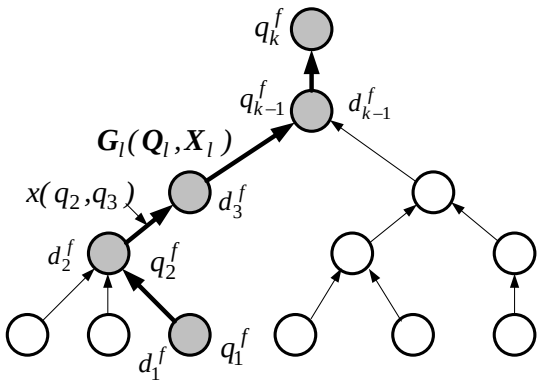

FIGURE III. THE THREAD $\mathbf{H}_f$ IN THE GRAPH $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ OF THE TASK Z_l

In other words, a thread is a certain sequence of subtasks of the task Z_l, in which each next subtask takes results of the previous subtask as the initial data.

It is obvious that subtasks assigned to the vertices of the thread $\mathbf{H}_f$ can be executed only sequentially. The length T_f

of the thread $\mathbf{H}_f$ is the total time required for its solution, determined by

$$T_f = \sum_{i=1}^{k} \left(T(O_i) + T_{i,i+1} \right) \tag{1}$$

where $T(O_i)$ is the solution time of the subtask O_i, assigned to the vertex $q_i^f \in H_f$ $(i = 1,2,...,k)$;

$T_{i,i+1}$ is the time taken for transfer of the results of the subtask O_i to the resource which solves the next subtask O_{i+1}, assigned to the vertex $q_{i+1}^f \in \mathbf{H}_f$. Besides,

$$T(O_i) = \frac{v_i}{S_p(O_i)} \tag{2}$$

where v_i is the computational complexity of the subtask O_i;

$S_p(O_i)$ is the real performance of the resource R_p, which solves the subtask O_i; Besides,

$$T_{i,i+1} = \begin{cases} 0, \text{ if the subtasks } O_i \text{ and } O_{i+1} \text{ are solved by the same resource } R_p \in \mathbf{R}, \\ \dfrac{d_{r,r+1}^f}{Y_p}, \text{ if the subtasks } O_i \text{ and } O_{i+1} \text{ are solved by different resources } R_p \text{ and } R_c \end{cases} \tag{3}$$

Where $d_{i,i+1}^f$ is the size of data transferred from the subtask O_i to the subtask O_{i+1};

Y_p is the bandwidth capacity of the channel of the resource.

So, if the whole thread $\mathbf{H}_f$ is executed by one and the same resource R_p, then its length is by

$$T_f = \sum_{i=1}^{k} \frac{v_i}{S_p(O_i)} + \frac{d_{k,k+1}^f}{Y_p} \tag{4}$$

where $d_{k,k+1}^f$ is the size of data assigned to an outgoing (target) edge of the thread $\mathbf{H}_f$ (see Figure III).

It is obvious that the total execution time of the task $Z_l \in \mathbf{Z}$ will, first of all, depends on the length of the critical path of the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, i.e. of the thread with the maximum total computational complexity of its vertices [13], [14].

The authors of [8], [9] have suggested a scheme of multi-agent organization of the CCE dispatcher. In the scheme each resource R_j $(i = 1,2,...,N)$ has its program agent AR_i, which searches tasks for its resource R_j. So, the agent AR_i asks regularly a special Internet resource called "a bulletin board" (BB), where Customers place their tasks in a graph form. If the agent AR_j finds some task $Z_l \in \mathbf{Z}$ on the BB, it tries to join the society $\mathbf{R}_l \subseteq \mathbf{R}$ and to execute the longest thread which it is able to execute with the help of its resource R_j by the time specified by Customer. However, it is impossible to use such scheme of organization of the multi-agent CCE dispatcher, if the execution time for the tasks from the set $\mathbf{Z}$ is not specified, and they have to be solved during the minimum possible time.

Therefore, in contrast to the scheme, considered in [8], [9], besides the agents AR_i $(i = 1,2,...,N)$, which represent various resources $R_j \in \mathbf{R}$ $(i = 1,2,...,N)$, we suggest to include additional agents into the multi-agent CCE dispatcher: task agents AZ_j $(j = 1,2,...,M)$ or so-called "customer representatives" are responsible for minimization of the total execution time of the task $Z_l \subseteq \mathbf{Z}$ $(j = 1,2,...,M)$ (see Figure IV).

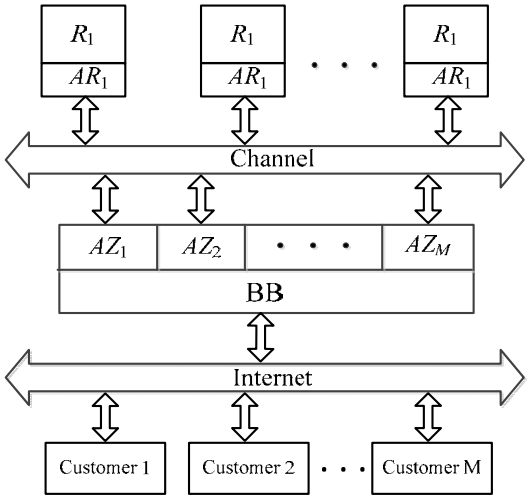

FIGURE IV. A STRUCTURE OF A CCE WITH MULTI-AGENT DISPATCHER

The aim of the multi-agent CCE dispatcher is to minimize the execution time of a task flow, coming from different Customers in arbitrary time moments. So, it is possible to describe functioning of the multi-agent CCE dispatcher as follows.

1) Customer generates its task $Z_l \in \mathbf{Z}$ in a form of a graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ and places it on the BB:

2) Each task $Z_l \in \mathbf{Z}$, placed on the BB, receives its agent AZ_l:

3) With the help of algorithms of critical path search [13], the agent AZ_l of the task Z_l selects a thread with the highest computational complexity $\mathbf{H}_1 = < q_1^1, q_2^1, ..., q_k^1 >$ in the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$: For this thread the value $v_1 = \left(\sum_{i=1}^{k} v_i^1 \right)$ is the highest; here v_i^1 $(j = 1,2,...,k)$ is the computational complexity of the subtask O_i, assigned to the vertex $q_i^1 \in \mathbf{H}_1$. In addition, the agent AZ_l specifies the possible (required) execution start time $t_s^1 = t_{curr}$, where t_{curr} is the current time. After that the thread $\mathbf{H}_1$ is placed on the BB for execution.

4) Each agent AR_j of the resource R_j $(i=1,2,...,N)$ in its turn asks the BB and tries to find a task for the resource R_j:

5) If the agent AR_j has found the thread $\mathbf{H}_1$ on the BB, placed for execution by the agent AZ_l of the task Z_l, then it estimates possibility of its own participation in execution of the task:

6) For this the agent AR_j selects a subthread $\mathbf{H}^1_{1j} =< q^1_1, q^1_2, ..., q^1_b > \subseteq \mathbf{H}_1$ $(b \le k)$ from the thread $\mathbf{H}_1 =< q^1_1, q^1_2, ..., q^1_k >$: The subtasks of the set $\mathbf{O}_i$, i.e. the subtasks executed by the resource R_j (see Fig.5), are assigned to the vertices of the subthread $\mathbf{H}^1_{1j}$. Besides, the agent AR_j specifies the time t^1_{sj}, when it will be able to start execution of the thread $\mathbf{H}^1_{1j}$ (i.e. when it will be free of execution of its previous subtasks), and the time t^1_{fj}, when execution of the subthread $\mathbf{H}^1_{1j}$ will be finished:

$$t^1_{fj} = t^1_{sj} + \sum_{i=1}^{b} \frac{v^1_i}{S_j(O_i)} + \frac{d^1_{b,b+1}}{Y_j} \tag{5}$$

where v^1_i is the computational complexity of the subtask O_i, assigned to the vertex $q^1_i \in \mathbf{H}^1_{1j}$ $(i=1,2,...,b)$;

$S_j(O_i)$ is the performance of the resource R_j during execution of the subtask O_i;

$d^1_{b,b+1}$ is the size of data assigned to the target edge $x(q^1_b, q^1_{b+1})$ of the subthread $\mathbf{H}^1_{1j}$;

Y_j is the channel bandwidth capacity of the resource R_j.

So, the selected subthread $\mathbf{H}^1_{1j}$ with its specified execution start time t^1_{sj} and execution completion time t^1_{fj} is a so-called "proposition" to participate in execution of the thread $\mathbf{H}_1$ made by the agent AR_p to the agent AZ_l of the task Z_l.

7) When "propositions" to participate in execution of the thread $\mathbf{H}_1$ are formed by the agents of all resources R_j $(i=1,2,...,N)$ of the CCE, the agent AZ_l of the task Z_l has to choose the best proposition among all the received ones: Let us consider the subthread $\mathbf{H}^1_{1p}$, proposed by the agent AR_p and with the maximum value

$$E_p = \frac{\sum_{i=1}^{b} v^1_i}{t^1_{fp} - t^1_s} \tag{6}$$

is the best one. Here v^1_i is the computational complexity of the subtask O_i $(i=1,2,...,b)$, assigned to the vertex $q^1_i \in \mathbf{H}^1_{1p}$; t^1_s is the required execution start time of the thread $\mathbf{H}_1$, specified by the task agent AZ_l; t^1_{fp} is the time, when execution of the subthread $\mathbf{H}^1_{1p}$ is finished by the agent AR_p.

The last equation defines a certain relative computational complexity of the "proposition" from the agent AR_p per unit of time. It is obvious, that the larger is the value E_p, the more complex fragment of the thread $\mathbf{H}_1$ will be executed by the agent AR_p, starting in the required time t^1_s.

8) The agent AZ_l sends a message to the agent AR_p and informs it, that the subthread $\mathbf{H}^1_{1p}$ with the maximum value E_p is assigned to the agent AR_p: The agent AR_p joins the society $\mathbf{R}_l$ which executes the task Z_l. In the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ of the task Z_l each vertex q^1_j $(j=1,2,...,b)$ of the subthread $\mathbf{H}^1_{1p}$ gets the number p of the resource R_p assigned to this vertex, and the execution time t^1_j calculated as follows

$$t^1_j = t^1_{sp} + \sum_{i=1}^{j} \frac{v^1_i}{S_p(O_i)} \tag{7}$$

where v^1_i is the computational complexity of the subtask O_i assigned to the vertex q^1_i;

$S_p(O_i)$ is the performance of the resource R_p during execution of the subtask O_i.

9) Then the agent AZ_l of the task Z_l excludes the vertices $q^1_1, q^1_2, ..., q^1_b$ of the subthread $\mathbf{H}^1_{1p}$ from the thread $\mathbf{H}_1$, and a new (shortened) thread $\mathbf{H}^2_1 =< q^1_{b+1}, ..., q^1_k >$ is formed (see Figure V): The number of the agent AR_p is assigned to the first vertex q^1_{b+1} of this thread. The agent AR_p supplies the initial data for the execution of the vertex, and the possible (required) start time of its execution $t^2_s = t^1_{fp}$, where t^1_{fp} is the time when execution of the subthread $\mathbf{H}^1_{1p}$ is finished. The time t^1_{fp} is calculated by formula (5). Then the agent AZ_l places the thread $\mathbf{H}^2_1$ on the BB for execution.

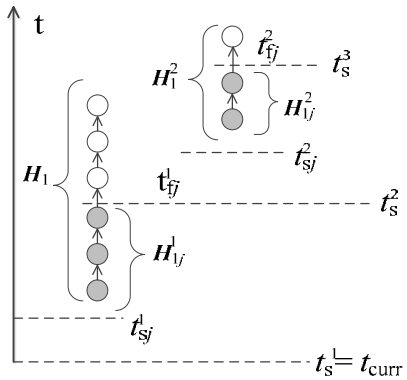

FIGURE V. DISTRIBUTION OF THE OPERATION OF THE THREAD $\mathbf{H}_1$ AMONG THE AGENTS AR_j $(j=1,2,...,N)$

10) Further, as described above, agents of various resources R_j $(i=1,2,...,N)$ estimate their capabilities of participation in execution of the thread $\mathbf{H}^2_1$: The agent AR_j $(j=1,2,...,N)$ selects the subthread $\mathbf{H}^2_{1j} =< q^1_{b+1}, ..., q^1_m > \subseteq \mathbf{H}^2_1$

$(m \le k)$ (see Figure V). The subtasks of the subset $\mathbf{O}_j \subseteq \mathbf{O}$ are executed by the resource R_j, and are assigned to the vertices of the subthread H^2_{1j}. Besides, the agent AR_j $(j = 1,2,...,N)$ specifies the execution start time t^2_{sj}, (i.e. the time, when it is able to start its execution, $t^2_{sj} \ge t^2_s$), and the execution completion time t^2_{fj} calculated as follows

$$t^2_{fj} = t^2_{sj} + \sum_{i=b+1}^{m} \frac{v^1_i}{S_j(O_i)} + \frac{d^1_{m,m+1}}{Y_j} \tag{8}$$

The agent AR_j sends the selected subthread $\mathbf{H}^2_{1j}$ to the agent AZ_l of the task Z_l as "a proposition" to participate in execution of the thread $\mathbf{H}^2_1$.

11) When the agent AZ_l of the task Z_l receives all "propositions" to participate in execution of the thread $\mathbf{H}^2_1$ from all agents AR_j $(j = 1,2,...,N)$, it accepts the best "proposition" from the agent AR_c, which suggests to execute the subthread $\mathbf{H}^2_{1c} =< q^1_{b+1}, q^1_{b+2}, ..., q^1_m >$ $(m \le k)$ with the maximum value:

$$E_c = \frac{\sum_{b+1}^{m} v^1_i}{t^2_{fc} - t^2_s} \tag{9}$$

The vertices of the subthread $\mathbf{H}^2_{1c}$ are assigned to the agent AR_c, which is informed about this fact with a special message, and the agent AR_p joins the society $\mathbf{R}_l$ to execute the task Z_l. Besides, each vertex q^1_j $(i = b+1,...,m)$ of the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ gets the number c of the resource R_c assigned to this vertex, and its execution time t^1_{jc} calculated as follows

$$t^1_j = t^2_{sc} + \sum_{i=b+1}^{j} \frac{v^1_i}{S_c(O_i)} \tag{10}$$

is the where v^1_i is the computational complexity of the subtask O_i assigned to the vertex q^1_i;

$S_c(O_i)$ is the performance of the resource R_c during execution of the subtask O_i.

12) Then the agent AZ_l forms a new thread $\mathbf{H}^3_1 =< q^1_{m+1}, ..., q^1_k >$, excluding the vertices of the subthread $\mathbf{H}^2_{1c}$ from the thread $\mathbf{H}^2_1$, i.e. $\mathbf{H}^3_1 = \mathbf{H}^2_1 / \mathbf{H}^2_{1c}$, and the first vertex q^1_{m+1} gets the possible (required) execution start time $t^3_s = t^2_{fc}$, where t^2_{fc} is the execution completion time of the subthread $\mathbf{H}^2_{1c}$, calculated by formula (8):

13) When the agent AZ_l of the task Z_l excludes the thread $\mathbf{H}_1$ from the task graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, a new graph $\mathbf{G}^1_l(\mathbf{Q}^1_l, \mathbf{X}^1_l) = \mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l) / \mathbf{H}_1$ (see Figure VI) is formed: In the graph $\mathbf{G}^1_l(\mathbf{Q}^1_l, \mathbf{X}^1_l)$ the agent AZ_l again selects the most computationally complex thread $\mathbf{H}_2 =< q^2_1, q^2_2, ..., q^2_r >$ (see Figure VI), which is placed for execution and is distributed among the agents of the resources R_j $(i = 1,2,...,N)$ just in a similar way as it was described above. As a result, all vertices of the thread $\mathbf{H}_2 =< q^2_1, q^2_2, ..., q^2_r >$ in the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ get numbers of the corresponding resources R_j assigned to the vertices, and their execution time t^2_f $(f = 1,2,...,r)$.

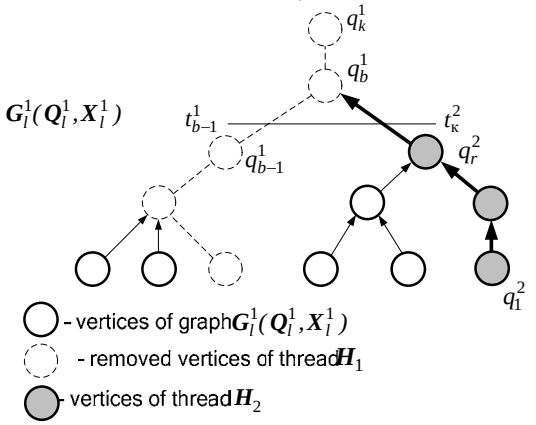

FIGURE VI. THE GRAPH $\mathbf{G}^1_l(\mathbf{Q}^1_l, \mathbf{X}^1_l)$ OF THE TASK Z_l MODIFIED BY THE AGENT AZ_l

14) Since the thread $\mathbf{H}_2 =< q^2_1, q^2_2, ..., q^2_r >$ is a branch of the thread $\mathbf{H}_1 =< q^1_1, q^1_2, ..., q^1_k >$ (i.e. the target vertex q^2_r of the thread $\mathbf{H}_2$ is incident with one of the vertices q^1_b of the thread $\mathbf{H}_1$) (see Figure VI), then it is necessary to check synchrony of the execution completion time of the thread $\mathbf{H}_2$ and the execution start time of the vertex q^1_b of the thread $\mathbf{H}_1$, incident with the thread $\mathbf{H}_2$: For this the agent AZ_l of the task Z_l compares the execution completion time t^2_f of the thread $\mathbf{H}_2$ calculated as follows

$$t^2_f = t^2_r + \frac{d^2_{r,r+1}}{Y_j} \tag{11}$$

where t^2_r is the time assigned to the target vertex q^2_r of the thread $\mathbf{H}_2$;

$d^2_{r,r+1}$ is the size of transferred data, assigned to the edge outgoing from the vertex q^2_r;

Y_j is the bandwidth capacity of the channel of the resource R_j, assigned to the vertex q^2_r; with the time t^1_{b-1} assigned to the vertex q^1_{b-1} of the thread $\mathbf{H}_1$. If $t^2_f > t^1_{b-1}$, then it means that the data, obtained as execution results of the subtasks of the thread $\mathbf{H}_2$, and required for execution of the subtask assigned to the vertex $q^1_b \in \mathbf{H}_1$, will be obtained later than the data, also required for solution of the subtask of the vertex q^1_b and obtained as execution results of the subtask of the vertex

q^1_{b-1} of the thread $\mathbf{H}_1$. In this case the agent AZ_l corrects the execution time schedule of the thread $\mathbf{H}_1$, increasing the time, assigned to its vertices $q_b, q_{b+1}, ...q_k$, by $\Delta t = t^2_{\text{f}} - t^1_{b-1}$.

15) After that the agent AZ_l of the task Z_l forms a new task graph $\mathbf{G}^2_l(\mathbf{Q}^2_l, \mathbf{X}^2_l) = \mathbf{G}^1_l(\mathbf{Q}^1_l, \mathbf{X}^1_l) / \mathbf{H}_2$, removing the vertices of the thread $\mathbf{H}_2$, and in the graph selects a new the most complex thread $\mathbf{H}_3$ and places it on the BB for execution: Distribution of the vertices (subtasks) of the task graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ continues until all its vertices (subtasks) are distributed, i.e. a new graph $\mathbf{G}^d_l(\mathbf{Q}^d_l, \mathbf{X}^d_l) = \varnothing$. As a result, each vertex $q_i \in \mathbf{Q}_l$ of the task graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ will get a corresponding number of resource R_j $(j = 1, 2, ..., N)$ for its execution, and the scheduled time t_i of its execution. In other words, an execution schedule of the task Z_l is formed.

16) After that the agent of the task Z_l informs Customer about the scheduled time t^k_l for execution of its task Z_l: The time t^k_l is calculated as follows

$$t^k_l = t_k + \frac{d_{k,k+1}}{Y_p} \tag{12}$$

where t_k is the time assigned to the target vertex q_k of the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$;

$d_{k,k+1}$ is the size of the result data assigned to the edge outgoing from the vertex q_k;

Y_p is the bandwidth capacity of the channel of the resource R_p assigned to the vertex q_k.

If Customer agrees, the agent AZ_l informs all agents of the resources R_j $(i = 1, 2, ..., N)$, involved into execution of the task (i.e. the members of the society $\mathbf{R}_l$), that it is necessary to execute their subtasks according to the time schedule.

17) When the agent AR_p of the resource R_p draws confirmation from the agent AZ_l of the task Z_l, that it is necessary to execute its own subthread $\mathbf{H}^m_f =< q^f_1, q^f_2, ..., q^f_r >$ of the task graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$, it adds the subtasks assigned to the vertices of the subthread $\mathbf{H}^m_f$ into its schedule.

18) At the execution start time t^f_{sp} of the subthread $\mathbf{H}^m_f$ the agent AR_p starts to execute the subtask sequence O^f_i $(i = 1, 2, ..., r)$, assigned to the vertices $q^f_i \in \mathbf{H}^m_f$: If execution of some subtask O^f_i requires initial data from other resources, then the agent AR_j check availability of the data. If the required initial data is still not available, then the agent AR_j goes to a standby mode.

19) When all required initial data is available the agent AR_p starts execution of the subtask O^f_i $(i = 1, 2, ..., r)$ with the help of its own resource R_p: When all subtasks O^f_i assigned to the vertices q^f_i $(i = 1, 2, ..., r)$ of the subthread $\mathbf{H}^m_f$ are executed, the agent AR_p informs the agent AZ_l of the task Z_l about successful execution of the subtasks of the subthread $\mathbf{H}^m_f$ and transfers the obtained results to the next resource R_p, which executes the adjacent vertex of the graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$.

20) After that the agent AZ_l checks if the schedule is fulfilled by the agent AR_p of the subtasks of the subthread $\mathbf{H}^m_f$: The agent AZ_l compares the scheduled time t^f_{f} of the subthread $\mathbf{H}^m_f$ completion with the current time t_{curr}. The time t^f_{f} is calculated as follows as follows

$$t^f_{\text{f}} = t^f_r + \frac{d^f_{r,r+1}}{Y_p} \tag{13}$$

where t^f_r is the time assigned to the target vertex q^f_r of the subthread $\mathbf{H}^m_f$;

$d^f_{r,r+1}$ is the size of transferred data assigned to the edge outgoing from the vertex q^f_r;

Y_p is the bandwidth capacity of the channel of the resource R_p;

If $t_{\text{curr}} > t_{\text{f}}$, then it means that a delay has occurred during execution of the task Z_l, and the agent AZ_l informs Customer about this delay. Besides, the agent AZ_l corrects the execution schedule for the next vertices $q_r, q_{r+1}, ..., q_k$ of the task graph $\mathbf{G}_l(\mathbf{Q}_l, \mathbf{X}_l)$ by the value $\Delta t = t_{\text{curr}} - t^f_{\text{f}}$ (see Figure VI).

21) When all subtasks of the task Z_l are executed (i.e. the task agent has confirmations from all members of the society $\mathbf{R}_l$, that their subthreads are successfully executed), the agent AZ_l informs Customer about successful solution of its task Z_l, and then the task Z_l is deleted from the BB: Application of the described principles of multi-agent dispatching of a heterogeneous CCE requires more detailed analysis and development of two algorithms such as the algorithm of the agent AZ_l of the task Z_l, and the algorithm of the agent AR_p of the computational resource R_j.

III. CONCLUSIONS

The text of your paper should be formatted as follows. To estimate the efficiency of the suggested methods and algorithms of CCE multi-agent dispatcher during solution of a flow of large-scale scientific problems, it is necessary to analyze functioning of the CCE in various modes. Due to the fact, that the simulated computational environment is a complex system, it is necessary to organize a targeted experiment for its complete research.

For the developed distributed CCE program model with multi-agent dispatcher we have designed a test stand, which contained 10 computational modules. Each module consisted of 1 to 16 processor cores, connected by a shared network with the bandwidth capacity up to 1000 Mbit per second. Taking into account capabilities of virtualization, the test stand has

provided simulation of a CCE operating with various values of the initial parameters such as:

- the size of resources of the CCE;
- the performance of resources during solution of subtasks of various types;
- the bandwidth of the channels, which connect resources with cloud infrastructure;

On the base of parameter combinations, formed earlier, we have performed sets of experiments for various groups of system parameters.

The research results have proved that the suggested method is efficient, the values of the parameter Y1 in various sets of experiments (even for large number of system nodes) remain high enough. The final average value of efficiency of executors is 71.53 for all sets of experiments.

So, in spite of the fact, that CCE dispatching during task solution is realized at the level of executors, loading of computational resources remains high enough. That is why it is possible to conclude that the developed methods and algorithms are effective from the system executors' point of view.

Research results has proved that the main advantage of the suggested multi-agent approach to resource dispatching in CCE is adaptation of computational process to computational capabilities of CCE resources. Besides, in comparison to traditional, centralized organization of the dispatcher of the cloud environment, in this case requirements to servers (bulletin boards) become more simple. Owing to this, penalty of cloud environment organization can be considerably reduced. As a result, cost of cloud computations can also be reduced, and process of cloud environment scaling can be simplified.

So, it is possible to conclude, that the suggested approach is usable for design of enterprise CCEs for solution of a flow of large-scale tasks. Owing to such CCEs it will be possible to increase considerably efficiency of computational equipment usage, and to increase flexibility and fault tolerance of computational process.

ACKNOWLEDGMENT

Research is being conducted due to financial support of Russian Ministry of Science and Higher Education, contract № 14.575.21.0152.

REFERENCES

[1] B. Kepes, "Understanding the cloud computing stack: Saas, PaaS, IaaS," Divers. Ltd., pp. 1–17, 2011.Podani,J. (1994) Multivariate Data Analysis in Ecology and Systematics. SPB Publishing, The Hague.

[2] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on, 2012, pp. 877–880.

[3] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, 2010, pp. 27–33.

[4] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (IAAS)," Int. J. Eng. Inf. Technol., vol. 2, no. 1, pp. 60–63, 2010.

[5] F. Tao, L. Zhang, V. C. Venkatesh, Y. Luo, and Y. Cheng, "Cloud manufacturing: a computing and service-oriented manufacturing model," Proc. Inst. Mech. Eng. Part B J. Eng. Manuf., vol. 225, no. 10, pp. 1969–1976, 2011.

[6] M. Armbrust et al., "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[7] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and others, "Cloud Computing," IEEE Netw., p. 4, 2011.

[8] A. Kaliaev, "Multiagent approach for building distributed adaptive computing system," Procedia Comput. Sci., vol. 18, pp. 2193–2202, 2013.

[9] A. I. Kalyaev and I. A. Kalyaev, "Method of multiagent scheduling of resources in cloud computing environments," J. Comput. Syst. Sci. Int., vol. 55, no. 2, pp. 211–221, 2016.

[10] H. Li and N. K. Womer, "Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming," Eur. J. Oper. Res., vol. 246, no. 1, pp. 20–33, 2015.

[11] S. Gill, I. Kockar, and G. W. Ault, "Dynamic optimal power flow for active distribution networks," IEEE Trans. Power Syst., vol. 29, no. 1, pp. 121–131, 2014.

[12] A. Kalyaev and I. Korovin, "Adaptive multiagent organization of the distributed computations," AASRI Procedia, vol. 6, pp. 49–58, 2014.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT press, 2009.

[14] E. M. Reingold, J. Nievergelt, and N. Deo, Combinatorial algorithms: theory and practice. Prentice Hall College Div, 1977.