Theoretical and Applied Informatics
ISSN 1896–5334
Vol.24 (2012), no. 4
pp. 271–291
DOI: 10.2478/v10179-012-0017-0

Optimization of Overlay Computing Systems with Many-to-Many Transmissions

KRZYSZTOF WALKOWIAK, ANDRZEJ KASPRZAK, KAROL ANDRUSIECZKO

Department of Systems and Computer Networks, Faculty of Electronics,
Wroclaw University of Technology,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
Krzysztof.Walkowiak@pwr.wroc.pl

Received 9 May 2012, Revised 28 September 2012, Accepted 6 October 2012.

Abstract: The problem that this paper investigates, namely, optimization of overlay computing systems, follows naturally from growing need for effective processing and consequently, fast development of various distributed systems. We consider an overlay-based computing system, i.e., a virtual computing system is deployed on the top of an existing physical network (e.g., Internet) providing connectivity between computing nodes. The main motivation behind the overlay concept is simple provision of network functionalities (e.g., diversity, flexibility, manageability) in a relatively cost-effective way as well as regardless of physical and logical structure of underlying networks. The workflow of tasks processed in the computing system assumes that there are many sources of input data and many destinations of output data, i.e., many-to-many transmissions are used in the system. The addressed optimization problem is formulated in the form of an ILP (Integer Linear Programing) model. Since the model is computationally demanding and NP-complete, besides the branch-and-bound algorithm included in the CPLEX solver, we propose additional cut inequalities. Moreover, we present and test two effective heuristic algorithms: tabu search and greedy. Both methods yield satisfactory results close to optimal.

Keywords: optimization, computing systems, overlay, heuristics

1. Introduction

Development of various IT systems commonly applied in almost all areas of human activity generates huge amount of data, which analysis needs dedicated, high-power computers. Therefore, in recent years distributed computing systems have been gaining much popularity in many both academia and industry to enable effective processing related to many areas, e.g., medical data analysis, collaborative visualization of large scientific databases, climate/weather modeling, bioinformatics, experimental data acquisition, financial modeling, earthquake simulation, astrophysics, etc. [15], [23], [24].

The distributed computing systems can be categorized as Grids using special dedicated high-speed networks [24] and overlay-based systems using the Internet as an underlying physical network [4], [18], [21]. In this work, we consider an overlay computing system. The motivation is that overlay systems provide considerable network functionalities (e.g., diversity, flexibility, manageability) in a relatively simple and cost-effective way. Moreover, the overlays can be deployed regardless of physical and logical structure of underlying networks. The considered computing system includes a set of computing elements spread geographically (e.g., clusters or other hardware equipment). The workflow of the system assumes that input data generated in a number of nodes is transmitted to computing nodes that is responsible for processing the data and finally to deliver the results to all destination nodes. It should be noted that such a workflow is generic and can model many real applications applied in distributed manner, e.g., image processing, data analysis, numerical computations. The criterion of the optimization process is to minimize the operational cost (OPEX) of the system embracing expenses related to two most important resources, i.e., processing equipment and network connections.

The main innovation of our work – comparing to previous works related to Grid optimization – is joint optimization of scheduling and network capacity in overlay-based systems with many-to-many transmissions. Note that most of previous papers concentrate on Grid optimization in the context of one-to-many transmissions in optical networks (e.g., [5], [11], [20], [22]) or focus only on scheduling optimization with one-to-many transmissions (e.g., [12]). It should be noted, that in this work we continue our research on optimization of distributed systems. In our previous work [12], we examined an optimization problem similar to the model formulated in this paper. The major modification of the model considered in this work is joint optimization of task scheduling and network capacity, while in [12] only the task allocation was analyzed, since network capacity was given.

The main contributions of the paper are as follows. First, based on a new architecture of the overlay computing system applying many-to-many transmissions, we formulate a novel ILP optimization model of the system. Second, for the considered problem we propose cut inequalities to improve performance of the branch-and-bound algorithm. Third, two effective heuristic algorithms are created. Fourth, we report results of numerical experiments run to examine effectiveness of developed heuristics as well as to evaluate performance of the computing system in terms of various parameters.

The rest of the paper is organized in the following way. In Section 2, we describe architecture of the considered overlay computing system, formulate a corresponding ILP optimization model and introduce cut inequalities. Section 3 includes description of two heuristics proposed to solve the problem. In Section 4, we present and discuss results of numerical experiments. Section 5 describes related works. Finally, the last section concludes this paper.

2. Optimization Model of Overlay Computing Systems

The optimization model presented in this section is constructed according to assumptions taken from real overlay computing systems and previous works. Moreover, the model is generic, i.e., various computing systems, e.g. Grids, public resource computing systems match to our model.

2.1. Assumptions

The considered distributed computing system is constructed as a set of computing nodes (individual computers or clusters) – indexed by $v = 1,2,\ldots,V$ – connected to the overlay network. Note that overlay networks provide much flexibility. Thus, many currently deployed network systems applies overlay approach, e.g., Skype, IPTV, Video on Demand, SETI@home, etc. As the considered computing system works on the top of an overlay network, computing nodes are connected by virtual links that in the underlying network are realized by a path consisting of physical links. According to [25], nodes' capacity constraints are typically sufficient in overlay networks. Moreover, in the concept of overlay networks usually the underlay physical network is assumed to be overprovisioned and the only bottlenecks are access links [12]. Therefore, the only network capacity constraints of the model are set on access links. As we consider a capacity and flow allocation problem, the access link capacity is to be dimensioned – the integer variable y_v denotes the number of capacity modules allocated to the access link of node v. We assume that each node v is assigned to a particular ISP (Internet Service Provider), which offers high speed access link a capacity module m_v given in Mbps (e.g., Fast Ethernet). For the sake of simplicity, we assume that the whole capacity of the access link is devoted to the considered computing system. However, the model can be easily modified to incorporate additional background traffic on each access link following from other types of services used at each node. The only required modification is to add system this additional background flow to the flow of the computing system before the link capacity is checked.

Each node v is assigned with a maximum limit on processing rate p_v, i.e., each node is already equipped with some computers and p_v denotes the number of uniform computational tasks that node v can calculate in one second. However, the problem can be easily modified to enable optimization also of this kind of resource by introducing additional variable to dimension processing resources.

The computing system is designed to process a set of computational projects $r = 1,2,\ldots,R$. Each project r is described by a set of parameters. The number of uniform computational tasks (of the same required processing power, e.g., a number of FLOPS) of project r is denoted as n_r. Each project is assigned with a set of source nodes that produce the input data and a set of destination nodes that are to receive the output data including results of computations. Constant $s(r, v)$ is 1, if node v is the source node of project r; 0 otherwise. In the same way, $t(r, v)$ is 1, if node v is the destination node of project r; 0 otherwise. In spite of the assumption that the uniform task for each project has the same computational requirement, the values of the input and output data transmit rate are specific for each computational project following from particular features of the project. Constant a_{rv} denotes the transmit rate of input data generated in node v and related to project v. If a particular node v is not the source node of project r, then $a_{rv} = 0$. Let $a_r = \sum_v a_{rv}$ denote the overall transmit rate of input data related to project r. Constant b_r defines the transmit rate of output data, respectively, per one task in project r. Constants a_{rv} and b_r are given in bps (bits per second).

The workflow of the computations is as follows. The input data of a particular task of project r is transferred from each source node to one or more computing nodes that process the data. Next, the output data (results of computations) is sent from the computing node to all destination nodes. We assume that the computational project is long-lived, i.e., it is established for a relatively long time (days, weeks). Consequently, the input and output data associated with the project is continuously generated and transmitted. Therefore, computational and network resources can be allocated in the system according to offline optimization. Moreover, we do not consider – as in many other works (e.g. [Nabrzyski] – the time dependency of each task (starting time, completion time, etc.). An integer variable x_{rv} denotes the number of project r tasks computed on node v. A corresponding auxiliary binary variable z_{rv} is 1 if node v calculates at least one task of project r, 0 otherwise.

Fig. 1 shows a simple example to illustrate the overlay computing system architecture addressed in this paper. The system contains five computing nodes denoted as A, B, C, D, and E. Two tasks are to be computed. Nodes A and E are the source node of each task. Therefore, they provide the input data related to the tasks (rectangles labeled i1A, i2A, i1E and i2E). Rectangles labeled p1 and p2 denote the places where a particular task is calculated. Rectangles labeled o1 and o2 denote results of computations related to tasks 1 and 2, respectively. Nodes B and D are destinations of each task. Moreover, we present network flows generated to deliver the input and output data. Solid lines denote the flow of input data. Circle with a number and node id inside shows the indices of tasks the input data is related to. Dotted line shows the flow of output data. Again, the numbers in red circles indicate task indices the data belongs to. Note that each task has two nodes providing the input data and two destination nodes, thus many-to-many transmission is applied.

We assume that the maximum number of computing nodes involved in one project cannot exceed a limit denoted by N. For instance, if $N = 1$, then all uniform task of each project must be computed only on one node. If we set $N = V$, the number of computing nodes is not limited. We refer to N as *split factor*. The motivation behind this assumption

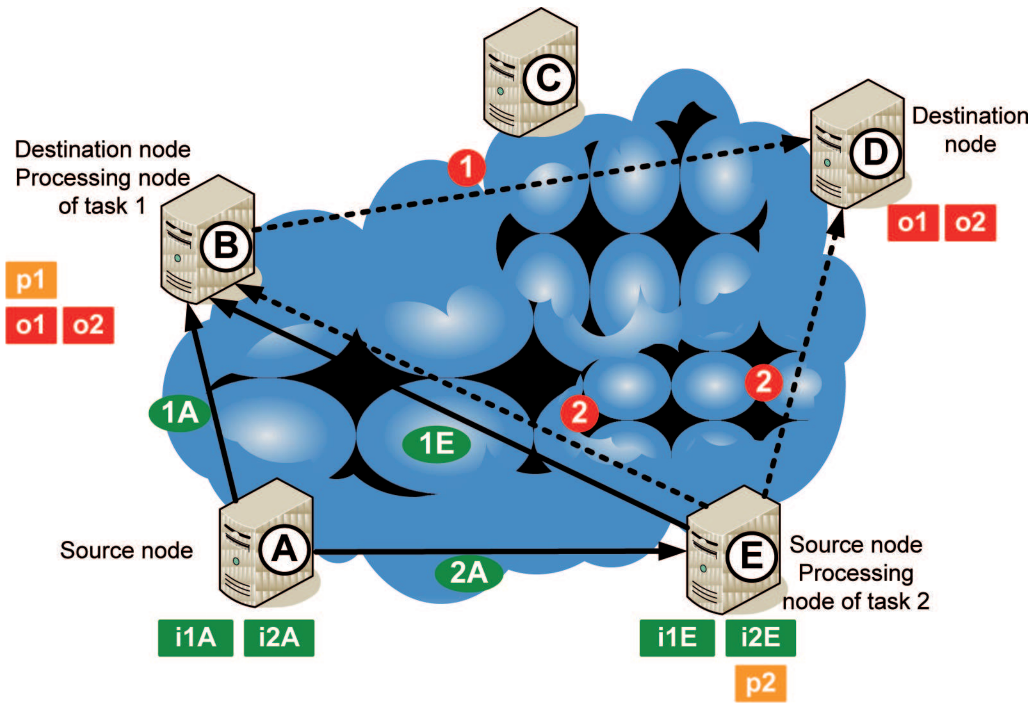

Fig. 1. Example of an overlay computing system

is related to management issues, i.e., less computing nodes (lower value of the split factor) facilitates the management of the distributed computing system. The objective of the optimization problem is to minimize the operating cost of the computing system including expenses related to the access link and processing of tasks. We dimension access links (variable y_v) and and optimize allocation of computing tasks (variable x_{rv}).

2.2. Objective Function

The objective function refers to operating costs (OPEX) of the system and includes two elements: networking costs and processing costs. The former element corresponds to costs of access links. Constant ξ_v given in euro/month denotes the whole OPEX cost related to one capacity module allocated for node v and includes leasing cost of the capacity module paid to the ISP as well as all other OPEX costs like energy, maintenance, administration. Since computing nodes are spread geographically in many countries with various ISPs, energy and maintenance costs, values of the module size and cost can be different for each node. Note that – if necessary – a number (more than 1) of capacity modules can be assigned to node v – thus the decision variable y_v is integer. For a good survey on various issues related to network costs see [16].

To model processing costs for each node v we are given constant ψ_v that denotes the OPEX cost related to processing of one uniform task in node v. The ψ_v cost is given in euro/month and includes all expenses necessary to process the uniform computational tasks (e.g., energy, maintenance, administration, hardware amortization etc.). Similarly to network costs ξ_v – the processing cost ψ_v is different for various nodes. This follows from the fact that nodes are placed in different countries with various costs related to energy, maintenance, administration. Various aspects of grid economics are discussed in [15].

2.3. Model

To formulate the problem we use the notation proposed in [16].

indices

$v,w = 1,2,\dots,V$	computing nodes
$r = 1,2,\dots,R$	projects

constants (additional)

p_v	maximum processing rate of node v (number of computational tasks that node v can calculate in one second)
n_r	size of project r (number of tasks in project)
a_{rv}	transmit rate of node v input data per one task in project r(Mbps)
a_r	transmit rate of overall input data per one task in project r(Mbps), $a_r = \sum_v a_{rv}$
b_r	transmit rate of output data per one task in project r(Mbps)
$s(r,v)$	$= 1$, if v is the source node of project r; 0, otherwise
$t(r,v)$	$= 1$, if v is the destination node of project r; 0, otherwise
t_r	number of destination nodes in project r
M	large numer
N	split ratio showing the maximum number of nodes involved in computing of one project
ψ_v	OPEX cost related to processing of one uniform task in node v (euro/month)
ξ_v	OPEX cost related to one capacity module of node v (euro/month)
m_v	size of the capacity module for node v (Mbps)

variables

x_{rv}	number of tasks of project r assigned to node v (non-negative integer)
y_v	capacity of access link for node $v$expressed in the number of modules (non-negative integer)
z_{rv}	$= 1$, if project r is calculated on node v; 0, otherwise (binary)

Objective

It is to find scheduling of task allocation and dimensioning of link capacity and minimize the OPEX cost related to access links selected for each node ($\sum_v\ y_v\ \xi_v$) and processing of computational tasks ($\sum_v \sum_r\ x_{rv}\psi_v$):

$$\min C = \sum_v y_v\ \xi_v + \sum_v \sum_r x_{rv}\psi_v \tag{1}$$

Constraints

a) The number of tasks assigned to each node cannot exceed the node's processing limit given by p_v.

$$\sum_r x_{rv} \le p_v v = 1, 2, \ldots, V \tag{2}$$

b) The flow downloaded by each node cannot be larger than the download capacity:

$$\sum_r (a_r - a_{rv})x_{rv} + \sum_{r:t(r,v)=1} b_r(n_r - x_{rv}) \le y_v m_v v = 1, 2, \ldots, V \tag{3}$$

In particular, the left-hand side of (3) denotes the flow entering node v and includes two terms. The former one ($\sum_r\ (a_r - a_{rv})x_{rv}$) is the transmit rate of input data of all projects calculated on node v. To process the task, node v must download all input data with transmit rate given by a_r. When the considered node v is the source node of project r, the incoming flow is decreased by a_{vr} denoting the input data available locally on the node. The latter term ($\sum_{r:t(r,v)=1} b_r(n_r - x_{rv})$) follows from the fact that each destination node v of the project r ($t(r,v)$ =1) must download the project results related to $(n_r - x_{rv})$ tasks (all tasks of project r except the tasks assigned to the node v). The right-hand side of (3) defines the download capacity of node v.

c) Analogously to (3), the upload capacity constraint must be satisfied:

$$\sum_{r:s(r,v)=1} a_{rv}(n_r - x_{rv}) + \sum_r (t_r - t(r,v))b_r x_{rv} \le y_v m_v v = 1, 2, \ldots, V \tag{4}$$

Again, the left-hand side of (4) compromises two elements. The first one ($\sum_{r:s(r,v)=1} a_{rv}(n_r - x_{rv})$) denotes that the source node v of project r ($s(r,v) = 1$) must upload its input data for computation of $(n_r - x_{rv})$ tasks (all tasks excluding the tasks assigned to node v). The second element ($\sum_r\ (t_r - t(r,v))b_r x_{rv}$) means that each node must upload the output data to all destination nodes of the project r given by t_r. Notice that we take into account the case when the considered node v is among destination nodes of the project r, therefore we use formula $(t_r - t(r,v))$ to denote the number of nodes to which the output data is transmitted.

d) All tasks of each project r = 1,2,...,R are assigned for processing:

$$\sum_v x_{rv} = n_r r = 1, 2, \ldots, R \tag{5}$$

The model given by (1)-(5) defines the basic version of the optimization problem. Additionally, we consider a limit on the maximum number of nodes involved in each project. The idea is to minimize the management overhead and reduce the number of nodes processing tasks related to the same project.

e) Binary variable z_{rv} is bound with decision variable x_{rv}:

$$x_{rv} \leq M z_{rv} r = 1, 2, \ldots, R v = 1, 2, \ldots, V \tag{6}$$

f) For each project r the number of nodes involved in the project cannot exceed the given threshold N:

$$\sum_v z_{rv} \leq N r = 1, 2, \ldots, R \tag{7}$$

2.4. Cut Inequalities

The problem (1)-(7) is NP-hard, since it is equivalent to the network design problem with modular link capacity [16]. Therefore, we propose to use additional cut inequalities that can be applied in construction of the branch-and-cut algorithm. Cut inequalities are introduced into the optimization problem to facilitate the branching phase and improve the quality of bounds. We consider the cut-and-branch variant of the B&C algorithm, in which cut inequalities are added to the root node of the solution tree. It means that all generated cuts are valid throughout the whole B&C tree [14].

The first cut is related to a lower bound on the variable y_v. Notice that if node v is a destination node of project r ($t(r, v) = 1$), it must receive the output data (results of computations) related to project r. Node v can receive the data in two ways: either as the input data (that is later processed on this node to obtain the output data) or as the output data. In the former case, two cases are to be considered. First, if node v is the source node of project r, then the download transmit rate is $(a_r - a_{rv})$, since a_{rv} is the rate of source date produced in node v. Second, when node v is not the source node of project r, then the download transmit rate is a_{rv}. Concluding, the download capacity of node v related to processing of task r must exceed the minimum of input and output data rates of project r. Let d_{rv} denote the lower bound of download flow related to node v and task r

$$d_{rv} = t(r, v) \min(s(r, v)(a_r - a_{rv}), (1 - s(r, v))a_r, b_r) \tag{8}$$

In analogous way, we analyze the upload capacity of node v related to processing of task r. If node v is the source node of task r ($s(r, v) = 1$), the data related to task r is to be delivered to all destination nodes of this task. Again, the data can be sent either as input data (rate a_{rv}) to another processing node or node v calculates task r and sends the output data (rate b_r) to $(t_r - t(r,v))$ nodes (all destination nodes except itself). The upload capacity of node v related to task r must exceed the following value

$$e_{rv} = s(r, v) \min(a_{rv}, ((t_r - t(r, v))b_r) \tag{9}$$

Thus, we can write the following constraints on the minimum capacity of each node v

$$\sum_r d_{rv} \leq z_v m_v v = 1, 2, \ldots, V \tag{10}$$

$$\sum_r e_{rv} \leq z_v m_v v = 1, 2, \ldots, V \tag{11}$$

Moreover, we can use the MIR (Mixed Integer Rounding) approach [9], [13] and rewrite the above constraints in the following way

$$[U + F8EE] \sum_r d_{rv}/m_v [U + F8F9] \leq z_v v = 1, 2, \ldots, V \tag{12}$$

$$[U + F8EE] (\sum_r e_{rv}/m_v [U + F8F9] \leq z_v v = 1, 2, \ldots, V \tag{13}$$

Next, we apply cuts based on the the Cover Inequality (CI) approach [1]. Two constraints are taken into account, i.e., the processing limit (3) and the split limit (7). To limit the number of possible cover inequalities, we first solve linear relaxation of the model (1)-(7). Next, in the obtained solution, we identify variables x_{rv} and z_{rv} that are not integer and for these variables the CI approach is applied in the context of constraints (3) and (7).

3. Heuristic Algorithms

In this section, we will present two heuristic developed to solve optimization model (1)-(7). The first method called AlgGreedy uses a simple greedy approach, while the second one called AlgTS is based on the tabu search method.

3.1. Greedy Algorithm

The pseudocode of the AlgGreedy method is shown in Fig. 2. The main idea of the algorithm is as follows. All individual tasks are processed in a single run of the algorithm (loop in lines 4-6). For each step of the loop, the best allocation of a task to a computing node is selected according to function MinCostAllocation() (line 5) and next the allocation is made according to function AllocateTask() (line 6). Note that the AllocateTask() function not only assigns a task of selected project $r\prime$ to the chosen computing node $v\prime$, but also updates the system, i.e., decreases the number of not allocated tasks of $r\prime$, decreases the available upload and download capacity of $v\prime$ according to transmit rates of output and input data in project $r\prime$. This means that the function MinCostAllocation() is always run in the current state of the system where performed task allocations limit the system resources. Finally, when all allocations are made (i.e., values of variable x_{rv} are selected), function AssignCapacitytoLinks() selects the smallest values of link capacity for each node (variable y_v) in order to satisfy the network flows generated by variables x_{rv}.

```
 1  GreedyAlg()
 2  begin
 3       $K = \sum_r n_r$ // number of all tasks taking into account all projects
 4       for $k = 1$ to $K$ // process all tasks of all projects
 5            ($r'$, $v'$) = MinCostAllocation() // id of the task and node
 6            AllocateTask($r'$, $v'$)
 7       AssignCapacitytoLinks()
 8  end
 9  MinCostAllocation()
10  begin
11       $c$ = MAX;
12       for $v = 1$ to $V$ // analyze all nodes for allocation
13            if (IsNodeFeasible($v$)) then
14                 for $r = 1$ to $R$ // analyze all projects
15                      if (IsProjectFeasible($r$)) then
16                           if (Cost($v$, $r$) < $c$) then
17                                $c$ = Cost($v$, $r$)
18                                $v' = v$
19                                $r' = r$
20       return ($k'$, $v'$)
21  End
```

Fig. 2. Pseudocode of AlgGreedy algorithm

The MinCostAllocation() function is described in lines 9-21. To find the best (with the lowest cost) allocation, all potential nodes are examined. However, only feasible nodes with free resources of the processing power (line 13) are considered as candidate nodes. Next, all feasible projects (with some not allocated tasks and satisfying the split ratio limit) are analyzed to find the allocation with minimum cost. Function Cost(v, r) (line 16) is a weighted sum of two elements. The first component is an average cost of allocation of one task of project r to node v taking into account the processing cost as well as the network cost. The network cost follows from the fact that allocation of the task to node v generates (i) flow of input data from the source node of the task to node v and (ii) flow of the output data from node v to all destination nodes of the task. For all these flows, network capacity is required. For instance, the cost related to the download capacity of the computing node v and one task of project r is estimated by formula $\xi_v a_r / m_v$. All other costs are calculated in analogous way. The second element of the cost estimates the additional costs (both processing and network) that would be required in the current system to serve the allocation of a new task of project r to node v. The additional processing cost is denoted by ψ_v. The additional network cost would be necessary only, when the allocation of one task of project r to node v generates the need to add a new capacity module(s) to some nodes (to provide enough capacity to send input/output data). Note that allocation of the new task to node v may not require expanding of existing capacity, since the already allocated resources may be sufficient to send all the required data.

3.2. Tabu Search Algorithm

The construction of the AlgTS method is based on the classical Tabu Search (TS) algorithm [7], [8]. The main idea of the TS method is to apply local searching procedure to avoid loops and local minimums. Short time memory is used to record recently taken moves and forbid them. In the following, we present the basic operations of the algorithm with a special focus on new elements. The solution is represented as a set of variables x_{rv} denoting allocation of tasks to computing nodes. Note that such a representation is sufficient to obtain to whole solution of the problem, since – similarly to AlgGreedy – the capacity dimensioning (variables y_v) is made on the base of network flows yielded by allocation of tasks. Moreover, values of variable z_{rv} can be directly obtained from variables x_{rv}.

The main element of the Tabu Search method is the neighbor search. In Fig. 3, we present the pseudocode of this procedure. The main idea is to move some tasks from one node to another. Note that the procedure is run on a particular solution of the problem including allocation of tasks given by variables x_{rv}). First, project $r\prime$ is selected at random (line 3) using RandGet function, which returns a randomly selected value among given values. After that, a potential source node w is selected at random among potential nodes involved in project $r\prime$, i.e., $z_{r\prime w} = 1$ (line 4). Next, there are two options of the method applied according $N_{r\prime}$ denoting the split value of the selected project $r\prime$. First, when $N_{r\prime} < N$ (i.e., the number of nodes involved in project $r\prime$ in the current solution is lower than the split ratio) we run the code in lines 6-10. In more detail, since the split ratio constraint is not achieved, we can select any destination node $w\prime$ different from w (line 7) and move m tasks from w to $w\prime$. Note that the number of moved nodes is selected at random (line 9) and cannot be larger than the number of project $r\prime$ tasks allocated to w (given by $x_{r\prime w}$) and the residual processing capacity of $w\prime$ (given by $p_{w\prime} - \sum_r x_{rw\prime}$). Function Move($r\prime$, w, $w\prime$,m) reallocates m tasks of project $r\prime$ from node w to w and updates values of variables accordingly (line 10). The second option is when $N_{r\prime} = N$ (i.e., the number of nodes involved in project $r\prime$ in the current solution is equal the split ratio) and lines 11-23 are run. Two procedures are possible to be excuted in this case, both with 50% probability. The first one – defined in lines 14-19 – assumes that all tasks from node w ($x_{r\prime w}$) are reallocated to a randomly selected node $w\prime$ that is not involved in project $r\prime(x_{r\prime w\prime} = 0)$ and additionally has enough residual processing power to process these new tasks ($p_{w\prime} - \sum_r x_{rw\prime} \geq \sum_r x_{r\prime w}$). If there are not such allocations, the algorithm moves automatically to the second procedure (lines 20-23). In this case, a randomly selected number of tasks is moved between two nodes involved in the considered project. However, again the number of reallocated tasks cannot exceed feasible limits related to the current number of tasks in node w ($x_{r\prime w}$) and residual processing capacity of node $w\prime$ ($p_{w\prime} - \sum_r x_{rw\prime}$).

The AlgTS algorithm works in a typical way. First, an initial solution is generated at random. Next, the searching process is started. Each iteration includes choosing the best solution from the neighbourhood and next moving to this solution. The aspiration is based on the best-fit criterion. Three parameters are used to tune the algorithm:

- *exploration_depth* limits the number of overall solutions analyzed in the neighborhood exploration;
- *exploration_range* denotes the number of start nodes used in the neighborhood exploration;
- *tabu_list_length* defines the length of the tabu list.

```
NeighborSearch()
Begin
    $r' = $ RandGet({$r$ : 1, 2,…,$R$})
    $w = $ RandGet({$v$ : 1, 2,…,$V \wedge z_{r'w} = 1$})
    $N_{r'} = \sum_v z_{r'v}$
    if $N_{r'} < N$ then
        $w' = $ RandGet({$v$ : 1, 2,…,$V \wedge w' \neq w$})
        $u = $ Min{$x_{r'w}$, $p_{w'} - \sum_r x_{rw'}$}
        $m = $ RandGet({1, 2,…,$u$})
        Move($r'$, $w$, $w'$,$m$)
    else
        if $N_{r'} = N$ then
            $t = $ RandGet({1, 2})
            if $t = 1$ then
                $w' = $ RandGet({$v$ : 1, 2,…,$V \wedge x_{r'v} = 0 \wedge p_v -$
                $\sum_r x_{rv} \geq x_{r'w}$})
                $m = x_{r'w}$
                if $w' \neq$ null then
                    Move($r'$, $w$, $w'$,$m$)
                    Exit
            $w' = $ RandGet({$v$ : 1, 2,…,$V \wedge x_{rv} > 0$})
            $u = $ Min{$x_{r'w}$, $p_{w'} - \sum_r x_{rw'}$}
            $m = $ RandGet({1, 2,…,$u$})
            Move($r'$, $w$, $w'$,$m$)
end
```

Fig. 3. Pseudocode of AlgTS algorithm

4. Results

Algorithms described in the previous section were implemented in C++ . Moreover, the ILP model presented in Section 2 as well as additional cut inequalities were implemented using CPLEX 11.0 solver [10]. The goal of experiments was threefold. First, we wanted to tune the AlgTS algorithm in order to find the best configuration of tuning parameters. Second, we compared results of heuristic methods against optimal results provided by CPLEX. Finally, we analysed performance of additional cut inequalities. Simulations were run on a PC computer with IntelCore i7 processor and 4GB RAM.

	Small systems
Number of computing nodes	20
Number of projects	10
Cost of capacity module	120-400
Processing cost of one unit	50-150
Processing limit of one node	10-40
Number of source nodes	1-4
Number of destination nodes	1-4
Input and output data rates	5-15

Tab. 1. Parameters of tested systems

27 unique configurations of distributed computing systems including 20 nodes and 10 projects were generated randomly. Table 1 presents information about ranges of system parameters applied in the random generation of the computing systems. For each unique configuration of nodes and projects 6 values of the split ratio (2, 4, 6, 8, 10 and 20) were tested, what gives 162 unique tests.

The first phase of simulations was devoted to tuning of the AlgTS algorithm. In Table 2, we report information about tuning parameters of the algorithm – for each parameter we present the tested values and the final selection of the parameter value according to performed experiments.

	Tested values	Selected value
exploration_depth	2,5,10,20,50,100,150,200,400	50
exploration_range	1,2,3,4,5,6,7,8,9,10	3
tabu_list_length	2, 5, 10, 15, 20, 15	10

Tab. 2. Tuning of the *exploration_depth* parameter

Now we present detailed analysis of parameters selection, which was performed in the context of small systems. In Fig. 4, we show performance of the AlgTS method as a function of the *exploration_depth*. The figure presents the gap to optimal results obtained for three values of the split ratio parameter. Note that other parameters were set to default values. We can easily notice that the best results are generated for *exploration_depth* equal to 50, 100 and 150. According to other experiments and to minimize the execution time of the algorithm, in further tests we set *exploration_depth* to 50.

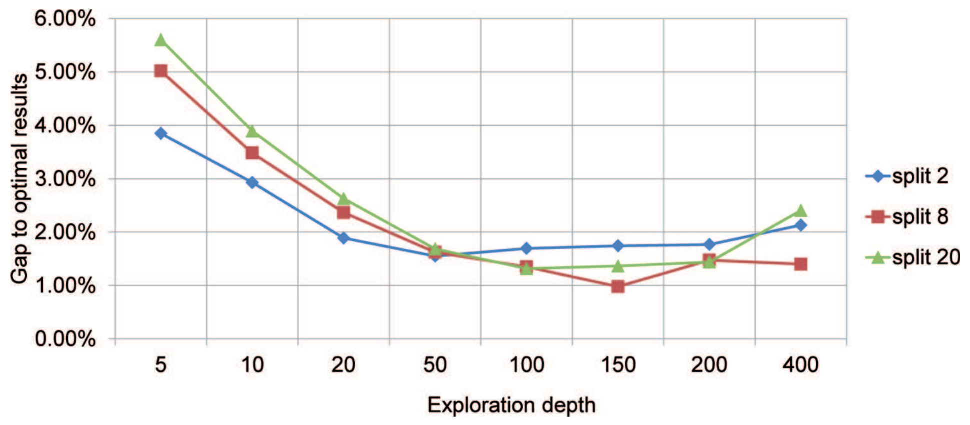

Fig. 4. Tuning of the *exploration_depth* paramter

The second tested parameter was *exploration_range*. In Fig. 5, we report results of the AlgTS according to all tested values of this parameter. Again, three values of the split ration were considered and other tuning parameters were set to default values shown in Table 2. The best tradeoff between quality of results and execution time is in our opinion reached for the case when *exploration_range* is set to 3.

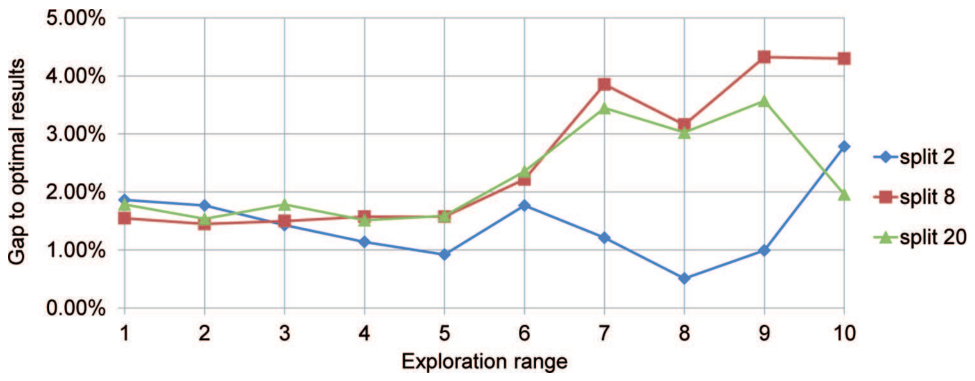

Fig. 5. Tuning of the *exploration_range* paramter

Finally, we examined the *tabu_list_length* parameter – Fig. 6 presents the results. The methodology was analogous as in two previous parameters. We decided to use for further experiments *tabu_list_length* equal to 10.

In Fig. 7, we show stability of the AlgTS method – for 4 different computing systems we executed the algorithm 20 times. We can watch that the algorithm provides quite stable results.

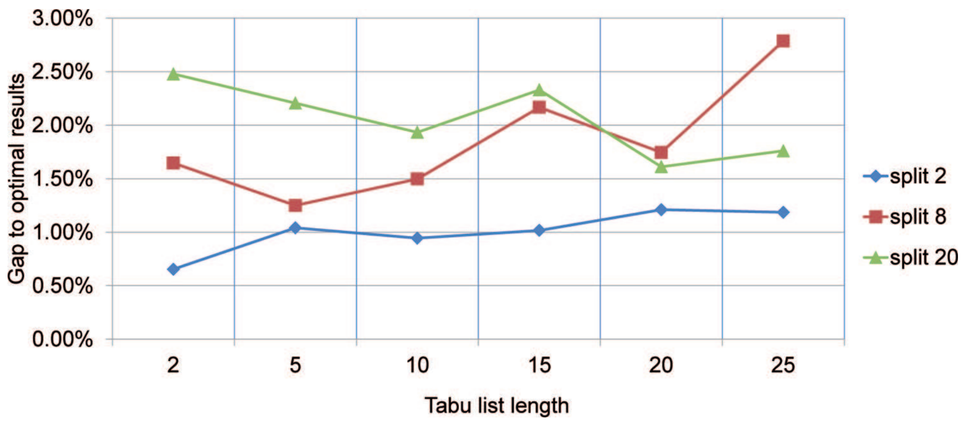

Fig. 6. Tuning of the *tabu_list_length* paramter

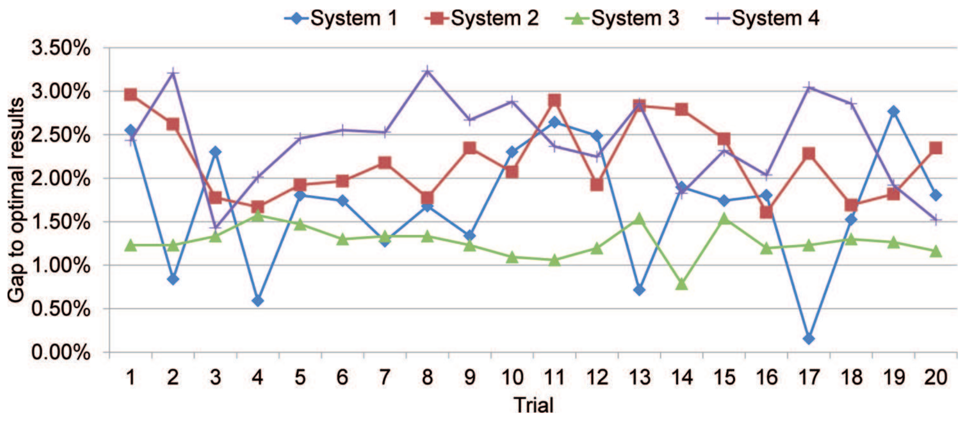

Fig. 7. Stability of AlgTS results

The second goal of experiments was devoted to comparison of heuristics against optimal results provided by CPLEX. In Table 3, we compare AlgGreedy and AlgTS against optimal results as a function of the split ratio. We report number of feasible results in each case, average gap to optimal results and length of 95% confidence intervals. First of all, we can see that AlgGreedy in the case of the lowest value of the split ration yields feasible results only in 1 of 27 cases. For larger values of the split ratio, AlgGreedy is more robust, however still there are some cases when this method does not provide feasible solutions. In contrast, AlgTS always yields feasible results. The average gap to optimal results over all tested networks is 8.40% and 1.68% for AlgGreedy and AlgTS, respectively. Relatively small values of 95% confidence intervals prove that both methods provide regular gaps to optimal results. The average execution time of tested methods is 0.03 seconds, 120 seconds and 140 seconds, for AlgGreedy, AlgTS, CPLEX, respectively.

	AlgGreedy			AlgTS		
Split ratio	Number of feasible results	Average gap to the optimal results	Lengths of 95% conf. intervals	Number of feasible results	Average gap to the optimal results	Lengths of 95% conf. intervals
2	1	3.34%	N/A	27	1.72%	0.25%
4	24	8.40%	0.74%	27	1.03%	0.15%
6	24	8.50%	0.80%	27	1.45%	0.22%
8	23	8.50%	0.84%	27	1.81%	0.22%
10	21	8.35%	0.86%	27	2.05%	0.29%
20	25	8.44%	0.76%	27	2.01%	0.27%

Tab. 3. Heuristics vs. CPLEX – result comparison and lengths of 95% confidence intervals

The next part of numerical experiments was devoted to examine the cut inequalities proposed in Section 2.4 to facilitate high computational time of the considered optimization problem. Recall that the main motivation behind the cut inequalities is to minimize the execution time of the exact algorithm producing optimal results. In Table 4, we show results of experiments conducted for 8 exemplary data sets with the split ratio set to 4. For each data set, we report the execution time and number of nodes in the branch and bound tree obtained in four case: without additional cuts, with MIR cuts, with CI cuts and with both kids of cuts. We can easily notice that the best performance offers the CI cut inequalities – on average the execution time is reduced by about 5%, while the number of B&B nodes is decreased by about 4%.

Data set	Execution time [s]				Number of B&B nodes			
	No Cut	MIR	CI	All	No Cut	MIR	CI	All
1	17737	17426	10046	14071	579	579	421	552
2	11638	11700	11824	11622	567	567	578	578
3	10420	10405	10530	10514	514	514	514	514
4	18892	18939	19079	19110	552	552	566	545
5	20046	20264	20218	30201	495	495	495	557
6	358	358	359	530	0	0	0	0
7	11372	11357	11856	14836	548	548	515	545
8	7113	7129	6988	6973	587	587	587	587

Tab. 4. Effectiveness of cut inequalities

5. Related Works

The authors of [20] focus on multicost algorithms for a joint scheduling of the communication and computation resources. They introduce a multi-cost scheme of polynomial complexity that provides reservations and selects the computation resource to execute the task and determines the path to route the input data. The results of numerical experiments show that in a Grid network in which tasks are either CPU- or data-intensive (or both), it is beneficial for the scheduling algorithm to jointly consider the computational and communication problems. The paper [22] addresses the optimization of optical Grids considering combined dimensioning and workload scheduling problem with additional survivability requirements. The Divisible Load Theory is applied to tackle the scalability problem and compare non-resilient lambda Grid dimensioning to the dimensions needed to survive single-resource failures. For regular network topologies, analytical bounds on the dimensioning cost are obtained. To validate these bounds, the authors report results of comparisons for the resulting Grid dimensions assuming a 2-tier Grid operation as a function of varying wavelength granularity, fiber/wavelength cost models, traffic demand asymmetry and Grid scheduling strategy for a specific set of optical transport networks. The authors of [5] present a dimensioning problem of computing Grids taking into account server location and capacity, network routing and capacity. They propose an integrated solution with joint optimization of the network and server capacity, and incorporate resiliency against both network and server failures. A case study on a meshed European network comprising 28 nodes and 41 links is presented. The results show that compared to classical (i.e., without relocation), they can offer resilience against both single link and network failures by adding about 55% extra server capacity, and 26% extra wavelengths. In [12], the authors consider optimization of overlay computing systems. The main goal is to allocate tasks to computing nodes in order to minimize the operational cost related to data transmission and processing. An ILP model is formulated and effective heuristic based on the GRASP method is proposed and evaluated. Paper [11] regards resilient optical Grids are considered. The authors examine how to maximize the grade of services for given transport capacities, while maximizing the protection level, i.e., against single link vs. single link and node (including server node) failures. A large scale optimization model is solved with help of Column Generation (CG) technique.

The idea of distributed computing systems have been gaining much popularity especially in the medical area, e.g., [2], [3], [6], [17], [19]. The BioGrid Australia platform enables authorized researchers a secure access to hundreds of thousands of privacy-protected health records provided by for BioGrid members and collaborators. Each data contributor maintains control of the access and use of their data. The platform includes Local Research Repository (LRR) servers located at collaborating institutions, which host replica copies of the institution's linked data sources. The Federated Data Integrator (FDI) server located at the BioGrid Australia is responsible to dynamically integrate data from linked data sources (via collaborating institution LRRs) in response to researchers' data queries. It should be noted that original data is not stored on the FDI. The next element of the system called Statistical Analysis System (SAS) is a collection of data interrogation, statistical analysis and reporting tools available through BioGrid

Australia to registered researchers. BioGrid Australia has enabled the implementation of many successful collaborative research projects, for a list of publications and other details see the website of the project [2]. MedlGrid is a proposal of a high performance, freely accessible medical image processing environment based on a distributed architecture. The idea of the system followed from a joined interaction between scientists devoted to the design and deployment of new and efficient tomographic reconstruction techniques, researchers in the field of distributed and parallel architectures, and physicians interested in experimenting with new advances in the field of image reconstruction and analysis. The major objective of the project was to design an easily accessible and usable platform providing medical experiments and research functionalities. In [3], the authors describe a prototypal grid architecture along with an open and distributed software environment. The computing infrastructure contains a storage server, a high performance parallel computing unit, and two PCs that act as clients to the system and that are located in geographically distant areas. The author of [6] introduce a model for developing and deploying a Self-Organized Map in an open source cluster and caching system under a popular distributed framework, J2EE. The main goal of the paper is to provide an efficient, flexible and low-cost model for implementing the SOM in a cluster environment. The major advantage of the proposed architecture is scalability, since any extra calculation work load required for a larger SOM can be accommodated easily by just adding more nodes or computers to the cluster. In [17] it is shown how the grid computing can be used to improve the operation of a medical image search system. The authors describe the basic principles of a content-based image retrieval (CBIR) system and identifies the computationally challenging tasks in the system. To tackle the most difficult issues of the content-based image retrieval process, an efficient architecture is introduced that applies a distributed grid computing approach to carry out the image processing in a distributed and efficient way. The authors of [19] report a project that designs and implements a pervasive health service infrastructure based on the grid system. Additionally, the proposed architecture includes P2P's resource sharing mechanism, to provide the personal health service. The personal health status is recorded, monitored, and mined in/from the proposed pervasive health service system for preventive medicine.

For more information related to Grid systems including description of representative examples of Grid computing projects refer to [2], [15], [23], [24]. More information on overlay systems can be found in [4], [18], [21].

6. Concluding remarks

This paper has detailed a novel optimization problem for overlay computing systems with many-to-many transmissions. We have formulated the problem in the form of the ILP model with additional cut inequalities proposed to facilitate complexity of the optimization problem. Next, we have developed two heuristic algorithms to solve the problem. Experimental results reported in the paper validate effectiveness of proposed heuristics – both methods provide satisfactory results in terms of the result quality and execution time. In more details, the greedy approach yields solution on average 8.40% worse than optimal ones, however the execution time is significantly lower comparing to the CPLEX solver. The Tabu Search method produces results only 1.68% lower comparing to optimal ones with lower execution time than the exact method.

Research results presented above have some practical implications. First, in the case of relatively stable and long-lived computational projects (e.g., time scale of days, weeks), the proposed optimization methods could be applied to find the most cost effective configuration of the system including both task scheduling and network dimensioning. In that case, the offline optimization is not an obstacle, as the system is to work for quite long time with the same configuration. Second, in the case of more dynamic systems, the results of offline optimization obtained using our methods can be used as a benchmark solution to evaluate quality of online optimization approaches. Finally, the proposed optimization framework containing both ILP model and heuristics could be used to examine main features of overlay computing systems with many-to-many transmissions regarding, i.e., how parameters like number of computing nodes, number of tasks, node processing rate, network capacity and others influence the operational cost of the system.

Encouraged by obtained results, in future work we plan to continue our research on distributed computing systems with many-to-many transmissions along two main research directions. On the one hand, we plan to introduce to the architecture additional survivability constraints in order to protect the system against network and computing device failures. Again, both exact and heuristic methods are planned to be developed to solve the problems. On the other hand, we are going to examine issues related to online optimization of overlay computing systems.

Acknowledgement

The work was supported by the Polish National Science Centre (NCN), under the grant N N519 650440.

References

1. C. Barnhart, C.A. Hane, P.H. Vance: *Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems*. Operations Research, Vol. 48, No. 2, pp. 318-326, 2000.

2. BioGrid Australia. http://www.biogrid.org.au/wps/portal.

3. P. Bonetto, G. Oliva, A.R. Formiconi: *MedIGrid: a medical imaging environment based on a grid computing infrastructure*. Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol.2, pp. 1338-1341, 2003.
4. J. Buford, H. Yu, E. Lua: *P2P Networking and Applications*. Morgan Kaufmann, 2009.
5. C. Develder *et al.*: *Survivable Optical Grid Dimensioning: Anycast Routing with Server and Network Failure Protection*. In Proc. IEEE International Conference on Communications, ICC 2011, pp. 1-5, 2011.
6. C.K. Fong: *A Study in Deploying Self-Organized Map (SOM) in an Open Source J2EE Cluster and Caching System*. IEEE/ICME International Conference on Complex Medical Engineering, 2007, pp. 778-781, 2007.
7. M. Gendreau, J. Potvin: *Handbook of metaheuristics*. Springer, 2010.
8. F. Glover: *Tabu Search - Part I*. ORSA J. on Computing, Vol. 1, No. 3, pp. 190-206, 1989.
9. O. Gunluk: *Branch-and-Cut Algorithm for Capacitated Network Design Problems*. Math. Programming, Vol. 86, No. 1, pp. 17-39, 1999.
10. ILOG: *CPLEX, 12.0 User's Manual*. France, 2007.
11. B. Jaumard, A. Shaikh: *Maximizing access to IT services on resilient optical grids*, 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), pp.1-6, 5-7 Oct. 2011.
12. T. Kacprzak, K. Walkowiak, M. Woźniak: *Optimization of Overlay Distributed Computing Systems for Multiple Classifier System – Heuristic Approach*. Logic Journal of IGPL, DOI: 10.1093/jigpal/jzr020, 2011.
13. H. Marchand, L. Wolsey: *Aggregation and mixed integer rounding to solve MIPs*. Operations Research, Vol. 49, No. 3, pp. 363-371, 2001.
14. J. Mitchell: *Branch-and-cut methods for combinatorial optimization problems*. in the Handbook of Applied Optimization, Oxford University Press, 2002.
15. J. Nabrzyski, J. Schopf, J. Węglarz (eds.): *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2004.
16. M. Pioro, D. Medhi: *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Publishers, 2004.
17. M.J. Pitkanen, Xin Zhou, A. Hyvarinen, H. Muller: *Using the Grid for Enhancing the Performance of a Medical Image Search Engine*. 21st IEEE International Symposium on Computer-Based Medical Systems, 2008. CBMS '08., pp. 367-372, 2008.
18. X. Shen, H. Yu, J. Buford, M. Akon (eds.): *Handbook of Peer-to-Peer Networking*. Springer, 2009.
19. S. Lu, K. Lai, D. Yang, M. Tsai, K. Li, Y. Chung: *Pervasive health service system: insights on the development of a grid-based personal health service system*. 2010 12th IEEE International Conference on e-Health Networking Applications and Services (Healthcom), pp. 61-67, 2010.

20. T. Stevens *et al.*: *Multi-Cost Job Routing and Scheduling in Optical Grid Networks.* Future Generation Computer Systems. Vol. 25, No. 8, pp. 912-925, 2009.

21. S. Tarkoma: *Overlay Networks: Toward Information Networking.* Auerbach Publications 2010.

22. P. Thysebaert *et al.*: *Scalable Dimensioning of Resilient Lambda Grids, Future Generation Computer Systems.* Vol. 24, No. 6, pp. 549-560, 2008.

23. A. Travostino, J. Mambretti, G. Karmous-Edwards (eds.): *Grid Networks Enabling Grids with Advanced Communication Technology.* Wiley, 2006.

24. B. Wilkinson: *Grid Computing: Techniques and Applications.* Chapman & Hall/CRC Computational Science 2009.

25. Y. Zhu, B. Li: Overlay Networks with Linear Capacity Constraints. *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 2, pp. 159-173, 2008.

Optymalizacja nakładkowych systemów obliczeniowych z transmisjami wielu do wielu

Streszczenie

Zagadnienia dotyczące optymalizacji systemów obliczeń rozproszonych zyskują w ostatnich latach na znaczeniu. Systemy obliczeń rozproszonych rozwijane są w dwóch podstawowych architekturach sieciowych. Po pierwsze, budowane są dedykowane sieci optyczne łączące ośrodki obliczeniowe. Po drugie, wykorzystuje się istniejącą infrastrukturę sieciową (np. Internet) dla budowania systemów pracujących w architekturze nakładkowej (ang. *overlay*). Ta druga koncepcja zyskuje ostatnią dużą popularność, gdyż umożliwia szybką i tanią realizację systemów obliczeniowych bez potrzeby mocnej współpracy z operatorami sieciowymi. W pracy rozważamy nakładkowy system obliczeniowy umożliwiający transmisje wielu do wielu – dane wejściowe do obliczeń są generowane w wielu źródłach (węzłach sieciowych), następnie po przetworzeniu są przesyłane do wielu odbiorców zainteresowanych wynikami obliczeń. W oparciu o zaproponowaną architekturę systemu, w pracy sformułowano problem optymalizacyjny mający na celu minimalizację kosztów operacyjnych systemu obejmujących koszty obliczeń i koszty przesyłania danych. Model zostałzapisany jako program całkowitoliczbowy. Z uwagi na fakt, że ten problem należy do klasy problemów NP-zupełnych, zaproponowano dodatkowe odcięcia dla algorytmu podziału i oszacowań oraz dwa efektywne algorytmy heurystyczne. Przeprowadzone eksperymenty obliczeniowe wykazały, że opracowane algorytmy dają wyniki bliskie optymalnym w mniejszym czasie niż algorytm optymalny zawarty w pakiecie CPLEX.