HEURISTIC ALGORITHMS FOR OPTIMIZATION OF TASK ALLOCATION AND RESULT DISTRIBUTION IN PEER-TO-PEER COMPUTING SYSTEMS

GRZEGORZ CHMAJ, KRZYSZTOF WALKOWIAK, MICHAŁ TARNAWSKI, MICHAŁ KUCHARZAK

Department of Systems and Computer Networks, Faculty of Electronics, Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
E-mail: grzegorz@chmaj.net, krzysztof.walkowiak@pwr.wroc.pl, mgtarnawski@gmail.com, michal.kucharzak@pwr.wroc.pl

Recently, distributed computing systems have been gaining much attention due to growing demand for various kinds of effective computations in both industry and academia. In this paper, we focus on Peer-to-Peer (P2P) computing systems, also called public-resource computing systems or global computing systems. P2P computing systems - in contrary to grids - use personal computers and other relatively simple electronic equipment (e.g., PlayStation console) to process sophisticated computational projects. A significant example of the P2P computing idea is BOINC (Berkeley Open Infrastructure for Network Computing) project. To improve performance of the computing system, we propose to use the P2P approach to distribute results of computational projects, i.e., result data is transmitted in the system like in P2P file sharing systems (e.g., BitTorrent). In this work, we concentrate on offline optimization of the P2P computing system including two elements: scheduling of computations and data distribution. The objective is to minimize the system OPEX cost related to data processing and data transmission. We formulate an Integer Linear Problem (ILP) to model the system and apply this formulation to obtain optimal results using the CPLEX solver. Next, we propose two heuristic algorithms that provide results very close to optimum and can be used for larger problem instances than those solvable by CPLEX or other ILP solvers.

Keywords: P2P computing system, distributed computing, optimization, heuristics, evolutionary algorithms

1. Introduction

Peer-to-Peer (P2P) paradigm appears to be an efficient alternative to client-server approach in emerging and future Internet applications. Moreover, numerous recent publications prove that the P2P concept has been gaining much attention, e.g., Christakidis et al. (2011), Couto da Silva et al. (2011), Kim et al. (2009), Ramzan et al. (2011), Suri et al. (2010), Terzo et al. (2011), Zhou et al. (2011). This follows mainly from the fact that in many cases the conventional client-server architecture faces various problems and limitations, including: scalability (i.e., growing number of users), diversity of new network applications, cooperation between many domains using various physical technologies, etc. On the other hand, virtualization including P2P systems, provides considerable network functionalities (e.g., diversity, flexibility, manageability) in a relatively simple way and regardless of physical and logical structure of underlying networks. Accordingly, various distributed applications employ the P2P and other network virtualization concepts, e.g., Christakidis et al. (2011), Couto da Silva et al. (2011), Kim et al. (2009), Ramzan et al. (2011), Suri et al. (2010), Shen et al. (2009), Zhou et al. (2011).

Another important trend that can be noticed in the current Internet is the growing demand for various kinds of effective computations, as both business and academia require huge computation power. In this work, we study issues related to Peer-to-Peer (P2P) computing systems called also public-resource computing system or global computing system.

1.1. Motivations. The first motivation behind our work is the growing need to provide large computational systems that are able to process huge amounts of data. This fact follows mainly from development of various computer systems applied in many areas of human activity. Secondly, most of existing distributed computing systems use a centralized approach to distribute results
In Section 7, we report related works. Finally, concluding showing performance of proposed optimization methods.

In Section 6, we present comprehensive numerical results include two heuristic algorithms (greedy and evolutionary). we formulate an ILP model of the system. Sections 4 and 5 in-proposed P2P computing system. In Section 3, we for-ganized as follows. Section 2 describes an architecture of various parameters describing the system.

1.3. Paper overview. The remainder of the paper is or-ganized as follows. Section 2 describes an architecture of the proposed P2P computing system. Next, we formulate an ILP (Integer Linear Programming) optimization model that can be optimally solved by exact algorithms (e.g., branch and cut) included in solvers like CPLEX (ILOG, 2009). Since the considered problem is NP-hard, only relatively small problem instances can be solved by exact methods. There-fore, to solve larger problem instances we develop two heuristic algorithms: a greedy method and an evolution-ary approach. The next significant contribution of this paper refers to extensive, numerical experiments that were conducted to compare all three optimization methods and to evaluate the computing system performance regarding various parameters describing the system.

1.2. Our contributions. In this work, we address several issues related to P2P computing systems with a special focus on static optimization. Our contributions include the following. First, we precisely describe main ideas and assumptions of the proposed P2P computing system. Next, we formulate an ILP (Integer Linear Programming) optimization model that can be optimally solved by exact algorithms (e.g., branch and cut) included in solvers like CPLEX (ILOG, 2009). Since the considered problem is NP-hard, only relatively small problem instances can be solved by exact methods. Therefore, to solve larger problem instances we develop two heuristic algorithms: a greedy method and an evolutionary approach. The next significant contribution of this paper refers to extensive, numerical experiments that were conducted to compare all three optimization methods and to evaluate the computing system performance regarding various parameters describing the system.

2. System architecture

In this section, we will briefly present an architecture of the distributed computing system addressed in this work. The idea of the system was introduced in our recent paper (Chmaj and Walkowiak, 2010a). We assume that the considered distributed system is a P2P computing sys-tem. Note that in contrary to grids, P2P computing sys-tems are focused on the application of personal com-puters and other relatively simple electronic equipment (e.g., PlayStation console) instead of supercomputers and clusters. The idea of P2P computing systems is applied for instance in the BOINC (Berkeley Open Infrastructure for Network Computing) architecture (Anderson, 2004). BOINC is an open-source, software computing platform using volunteered resources (BOINC, 2011).

The distributed system is composed of many comput-ing nodes (e.g., personal computers) connected into one logical structure by the means of the Internet. Thus, the system works in an overlay mode (Tarkoma, 2010). The underlying transport network (e.g., Internet) provides direct connectivity between overlay nodes with required Quality of Service parameters. Each node participating in the computing system is attached to the overlay network by an access link with specified download and upload capacity expressed in bps. The computational project, which is to be processed in the system is divided into small parts called tasks or blocks (Samanta et al., 2001; Stutzbach et al., 2004; Nabrzyski et al., 2004; Munidger and Weber, 2004). Each task requires the same computational power. Different tasks can be calculated on different nodes.

In most of existing systems the distribution of comput-ational results is done by a central node (Anderson, 2004; Krauter et al., 2002; Milojicic et al., 2002; Nabrzyski et al., 2004; Shen et al., 2009; Travostino et al., 2006). After processing of the task, the comput-ing node sends back the final result to the central node, which next distributes the results to other nodes. In highly cooperative environments, in which many participants of the system request to receive all results (e.g., scientific projects), the central node can become a bottleneck of the result delivery process. Therefore, in our architecture we use a distributed scheme for result delivery and apply the P2P approach effectively applied in file sharing systems, e.g., BitTorrent, Cohen (2003). Partial results (individual blocks) are not sent back to the central node, but they are delivered between participating nodes directly. Similarly to the BitTorrent protocol (Cohen, 2003), our system ap-plies a special node called tracker. The goal of the tracker is twofold. Primary, the tracker is responsible for schedul-ing, i.e., it assigns individual task to computing nodes ac-cording to received requests. Next, the tracker maintains

...
and provisions a database providing updated information related to the current state of the system, e.g., location of calculated results. We do not use fully decentralized P2P architecture (e.g., DHT), since in our opinion the BitTorrent approach with some central elements (i.e., trackers) is sufficient for our purposes. However, the system can be easily modified to apply fully decentralized P2P architectures to provide delivery of results. Note that in contrast to our system, the BOINC architecture assumes that the result block is sent back to the central node, which collects all results for further processing and analysis. For more information on P2P system refer to Shen et al. (2009), Tarkoma (2010).

The workflow of the system is as follows. The input computational project is composed of uniform blocks called source blocks. Each source block is forwarded to a node in order to be processed (computed) according to a selected scheduling procedure, which is the goal of optimization addressed in this work. When the processing is completed, a node informs the tracker that the result block is available to other nodes. Next, each node requesting the result block tries to download the result block. The distribution of result blocks is based on the P2P idea, i.e., each node can act both as a client (downloading) or a server (uploading). Consequently, each node that obtains a result block informs the tracker about this fact and the location database is updated adequately.

We assume that the distributed system is highly collaborative, e.g., a scientific project including several universities and laboratories working on a similar topic. Therefore, we make two important assumptions related to the system. First, each participant (computing node) is interested to obtain all results - consequently, every result block must be delivered to each node. Second, each participant must be involved in the project and compute (process) at least one source block. The second assumption is partially motivated by the problem of selfish users encountered in P2P systems. Many P2P systems introduce special incentive mechanisms to overcome this problem, e.g., „tit for tat” rule in BitTorrent (Cohen, 2003; Shen et al., 2009; Tarkoma, 2010). The most famous distributed processing framework - BOINC (and all systems based on it) provides the functionality to compute the complex, but dividable task, so the overall goal is similar to our system. The key difference is, that proposed distributed system guarantees the result delivery to all system participants and uses decentralized architecture, whereas BOINC is a fully centralized system where only the managing node receives full result. BOINC resembles unicast flow, and as we will show later in that paper, proposed P2P flow provides better results for distributed computation system.

Fig. 1 shows a simple example illustrating the main idea of the system. The considered system contains 4 nodes and 5 blocks. To make it clear, we do not show the tracker node in the figure. Notice that source block \( b = 1 \) is allocated to node 1, source block \( b = 2 \) is allocated to node 2, source blocks \( b = 3 \) and \( b = 5 \) are assigned to node 3 and finally source block \( b = 5 \) is processed on node 4. Moreover, we show in the figure the delivery scheme of each result block. For instance, block \( b = 1 \) is sent from node 1 to node 3, which uploads the block to nodes 2 and 4.

In the following sections, we show how to optimize performance of the presented distributed system. As the performance metric we propose to use the operating cost (OPEX) of the system including two elements: processing (computation) cost of each source block and transfer cost related to the requirement to delivery each result block to requesting nodes. The processing cost consists of expenditures related various aspects of the IT infrastructure (e.g., energy, maintenance, hardware amortization, etc.). The reader interested in more information related to economics of distributed computing systems is referred to Nabrzyski et al. (2004). The second part of the OPEX cost
is the delivery cost related to data (blocks) transfer in the network. This element incorporates all of network expenditures (e.g., lease cost of the access link, energy, maintenance, hardware amortization, etc.). Various aspects of network costs are addressed by Kasprzak (2001), Pioro and Medhi (2004).

As mentioned above, the proposed architecture is dedicated to a highly collaborative environment, where all participants are interested to receive results of computations. In consequence, the delivery of final results can generate high network traffic. Therefore, we propose a distributed dissemination of final results. It should be noted that many other architectures of distributed systems have been proposed in the recent years. For instance, Google in 2004 introduced a map-reduce framework (González-Velez and Kontagora, 2011). The inspiration comes from the map and reduce functions commonly applied in functional programing. In the map operation, the master node partitions original computing problems into smaller sub-problems and distribute it to worker nodes. After processing, each worker node passes the answer (results) back to the master node. Then, the reduce step is applied, i.e., the master node collects the answers to combine the solution of original problem. The main difference between our architecture and the map-reduce framework is final results delivery process. In order to reduce costs of network flows, computation results are sent directly to requesting nodes that combine by themselves the final results. While, in the map-reduce architecture, master nodes are responsible to combine final results. However, if we assume that each network node is a master node, both architectures provide similar operation.

3. Integer linear programming modeling

In this section, we formulate an ILP model of the distributed computing system described in the previous section. Note that ILP modeling is a research approach widely applied in many areas to evaluate and optimize various kinds of systems, e.g., computer networks (Kasprzak, 2001; Pioro and Medhi, 2004).

3.1. Assumptions. The considered system includes nodes (computing elements) indexed \( v = 1, 2, \ldots, V \). The tracker node is assumed to be overprovisioned, i.e., its processing and network resources are large enough to eliminate any bottlenecks and congestions. Moreover, the signaling traffic to/from the tracker node is relatively small comparing to transfer of result blocks. Consequently, to reduce the complexity of the ILP model, the tracker node is not included in the model. There are computational tasks (blocks) indexed \( b = 1, 2, \ldots, B \). We use the same index to refer to both source and result blocks. All blocks are of uniform size and the same processing requirement. The largest challenge in the model is the dynamic time scale of the system. Since we use the P2P approach for data delivery, the system changes with the elapsing time as new result blocks are downloaded by subsequent nodes. Most of previous research on network optimization assumes constant rate of transported data, e.g., Kasprzak (2001), Pioro and Medhi (2004). In our model, we assume that the time scale is divided into time slots (iterations) of the same size indexed \( t = 1, 2, \ldots, T \). In each iteration nodes may transfer blocks, but the information on new location of result blocks is updated at the end of the iteration. This means that block \( b \) may be downloaded in iteration \( t \) only from nodes, which posses that block at the start of iteration \( t \). The transfer of all blocks must be completed within \( T \) time slots. The same approach to modeling the time scale of P2P systems was applied in many papers, e.g., Arthur and Panigrahy (2006), Ganesan and Seshadri (2005), Munidger and Weber (2004), Miller and Wolisz (2011), Yang and de Veciana (2004).

According to our assumptions, each node \( v \) has a limited processing power \( p_v \), limited upload and download capacity \( u_v \) and \( d_v \), respectively. To simplify the model, we assume that \( p_v \) is expressed in blocks per time slot. Similarly, upload and download limits \( u_v \) and \( d_v \) are given in blocks per time slot.

The objective function denoting the OPEX cost compromises two kinds of costs: processing and transfer. To model the former element, for each node we are given constant \( c_v \) that denotes processing cost of one source block in node \( v \). In the context of the transfer costs, for each pair of nodes \( w \) and \( v \) constant \( k_{vw} \) denotes the cost of one result block transfer between these nodes.

To reduce complexity of the model, we assume that in time slot \( t = 0 \) all source blocks are computed and in the next iteration \( t = 1 \), the transfer of result blocks is initiated. Thus, delivery of source blocks to computing nodes is not included in the time scale of the system. However, costs related to transfer of source blocks to computing nodes are incorporated in the processing cost \( c_v \).

The model contains two kinds of decision variables. The first binary variable \( x_{bv} \) denotes allocation of source blocks to computing nodes and is 1, if source block \( b \) is assigned to node \( v \). The second binary variable \( y_{bwvt} \) denotes P2P transfers in the network, \( y_{bwvt} \) is 1, if result block \( b \) is transferred from node \( w \) to node \( v \) in iteration \( t \).

3.2. Model.

Indices

\[
\begin{align*}
 b &= 1, 2, \ldots, B, \text{blocks (source and result)} \\
 t &= 1, 2, \ldots, T, \text{time slots (iterations)} \\
 v, w, s &= 1, 2, \ldots, V, \text{node (peer)}
\end{align*}
\]

Constants

\( c_v \) cost of processing of one source block in node \( v \)
Heuristic Algorithms for Optimization of Task Allocation and Result Distribution ...
Moreover, notice that - according to model (1)-(8) - the number of all blocks in the system must be at least \( \sum_v a_v \). If some blocks are still not assigned to computing nodes (i.e., \( \sum_v a_v < B \)), we continue the procedure in a loop. First, for each node we create a following scoring metric

\[
e_v = mc_v + \sum_w k_{vw},
\]

that defines the cost of node \( v \) combining the processing cost \( (c_v) \) and the transfer cost to other nodes \( (\sum_w k_{vw}) \). Note that \( m \) is a tuning parameter to find the tradeoff between processing and transfer costs. In next steps, we normalize value of the \( e_v \) metric and calculate parameter \( g_v \)

\[
g_v = \begin{cases} 
  0 & \text{when } p_v = a_v \\
  \frac{e_{max} - e_v}{e_{max}} & \text{otherwise.}
\end{cases}
\]

Finally, we allocate remaining blocks to subsequent nodes sorted according to decreasing value of \( g_v \). Following condition 4 each node is allocated with \( p_v - a_v \) source blocks. Note that the last processed block can receive less than \( p_v - a_v \) blocks.

The second algorithm PH2 yields the configuration of block transfer (variables \( y_{bvw} \)). The algorithm has one input parameter \( q \) that denotes the maximum number of block transfer in a single time slot. The idea behind the PH2 algorithm is as follows. In each time slot \( t < T \) we make at most \( q \) transfers. All remaining transfers are tried to be made in the last iteration. Since the objective is to minimize the transfer cost, we attempt to make as many transfers as possible between nodes with low values of the transfer cost \( k_{vw} \). Therefore, we sort all node pairs \( (w, v) \) in ascending order of the \( k_{vw} \) metric. Next, one by one we allocate to each analyzed node pair as many as possible transfers in both directions, as the \( k_{vw} \) cost is assumed to be symmetric. All necessary constraints of the problem are checked (e.g., node capacity). In the last iteration, all remaining transfers are arranged. Input parameter \( q \) is used to find the tradeoff between the objective of the optimization (transfer cost) and problem constraints (the transfer must be completed in a particular number of iterations and every node is limited by download and upload capacity limit).

Below we present algorithm PHA that combines two heuristics described above to solve problem (1)-(8).

**Algorithm PHA**

The algorithm uses the following input parameters: \( j_m \) - number of algorithm’s iterations, \( j_m < 30 \).

**Step 1.** Set \( q = \left\lfloor \frac{B(V-1)}{T} \right\rfloor \)

**Step 2.** For the following values of the \( m \) parameter \( m = 1, m = \lfloor B/V \rfloor, m = \lfloor B/2 \rfloor \) run PH1 (passing subsequent parameters \( m \)) and PH2 (passing parameter \( q \)). If for all tested values of parameter \( m \) no feasible solution was obtained stop the algorithm. Otherwise, let \( m^* \) denote the value of parameter \( m \), for which the best feasible solution was found. Let \( C_j \) denote the best value of the cost function.

**Step 3.** Calculate \( g_s \) according to the following formula:

\[
g_s = \left[ \frac{30 - j_m}{29} (q - 1) + 1 \right].
\]

Set \( j = 1 \) and \( q = q + g_s \).

**Step 4.** Run PH1 (passing parameter \( m^* \)) and PH2 (passing parameter \( q \)). If a feasible solution is found, let \( C_j \) denote the obtained cost. Otherwise, if there is no feasible solution set \( C_j = \infty \).

**Step 5.** Set \( j = j + 1 \) and \( q = q + g_s \). If \( q \leq \sum_v u_v \) and \( j \leq j_m \) then go to Step 4. Otherwise, go to Step 6.

**Step 6.** Stop the algorithm. Result of PHA is \( C^* = \min_{i=0,1,2,..,j_m-1}(C_j) \).

In the first phase of the algorithm (Steps 1-2) we examine three values of parameter \( m \) and select for further tests a value that provides the lowest value of the objective function. In the second phase (Steps 4-5) we focus on the parameter \( q \). In each iteration, we increase value of \( q \) by \( g_s \) defined in (12). We continue this process until the iteration count \( j \) does exceed iteration limit \( j_m \). The second stopping condition is related to parameter \( q \), i.e., if \( q > \sum_v u_v \) the algorithm stops, since according to the construct of algorithm PH2 in such case there is no chance to improve the result. Finally, the algorithm returns the best found value of the cost function.

5. **Evolutionary algorithm**

In this section, we introduce a new algorithm named Peer-to-Peer Evolutionary Algorithm (PEA) proposed to solve the optimization problem given by (1)-(8). The main idea of the PEA algorithm follows from evolutionary methods (Michalewicz, 1996). Various optimization problems have been successfully attacked by hybrid intelligence including evolutionary methods, e.g., Pioro and Medhi (2004), Przewozniczek et al. (2011), Wozniak (2009).

5.1. **Chromosome coding.** We assume that a single individual represents a vector of \( B \) (number of blocks) trees - each tree includes \( V \) nodes and denotes the transfer of each block. Root node \( v \) of the tree associated with block \( b \) defines the node selected to process the block \( b \), i.e., \( x_{vb} = 1 \). Thus, using the proposed representation we can easily obtain values of decision variables \( x_{vb} \) and \( y_{bvw} \). The representation guarantees that constraint (3) is satisfied, i.e., each source block is assigned for processing to exactly one node (root node of the tree). Moreover, condition (5) is also assured - each tree includes all nodes, thus each result block is delivered to each peer. To address constraint (2) guaranteeing that each peer must process at least one source block, the individual is divided
into two parts: $V$ trees (basic part) and $(V - B)$ trees (extension part). Remaining constraints of the problem (i.e., constraints (4) and (6)-(8)) are satisfied by algorithm’s operators (crossover and mutation). Fig. 2 shows a simple example of a chromosome coding for a network with $V = 4$ nodes and $B = 5$ blocks (the same example as in Fig. 1). The basic part contains 4 genes, the extension part includes the remaining block $b = 5$.

5.2. Algorithm overview. Fig. 3 presents the block diagram of the PEA algorithm. Now we will describe the most important elements of the algorithm.

The start population is generated according to the following heuristic. It is easy to notice that some of nodes are better for allocation than others. Factors that promote block allocation on a particular node $v$ are: low processing cost (constant $c_v$), low mean transfer cost to all other nodes ($k_v = \sum_w k_{vw}/(V - 1)$) and high upload limit of node (constant $u_v$). According to these observations, we define below a new constant $s_{nv}$ that denotes a heuristic score of node $v$:

$$s_{nv} = \frac{u_v}{c_v k_v}.$$  \hspace{1cm} (13)

The start population is generated with use of the roulette method - blocks are allocated on nodes with probability proportional to value of $s_{nv}$ [13]. According to our preliminary experiments, such a generation method provides significant improvements of the algorithm’s performance.

To generate subsequent populations, the PEA algorithm applies the tournament selection. The size of mating pool (i.e., number of individuals selected to perform the crossover expressed as a percentage of the population size) is given by an input tuning parameter called size of mating pool. Since the value of the size of mating pool parameter is below 100%, the remaining part of the next population is simply copied from the current population (using random selection). Moreover, a kind of elite promotion is applied, i.e., the best individual of the current population is also copied to the subsequent population. The algorithm is run for a given number of iterations, which is one of input parameters. The individual fitness function is calculated as the reciprocal of the individual cost given by (1). Since it was very hard to create single crossover operator providing reasonable performance, we decided to apply multiple evolutionary operators. All of those operators are used randomly with probability given as algorithm’s input parameters. Two crossover operators were created: tree exchange and path exchange. The tree exchange operator involves two parents (primary and secondary) and results in two children. Some of $B$ trees are exchanged between two individuals - the number of exchanged trees is determined by tree exchange probability given as a parameter. Only a tree that does not violate problem constraints is considered as a candidate tree. Additionally, for the basic part of the individual, only trees with the same root node as the primary tree are considered. In the tree election process we use the roulette method - each gene representing a tree of block $b$ and rooted at node $v$ is assigned with the following score:

$$s_{tb} = \frac{s_{nv}}{c_t b_t}.$$  \hspace{1cm} (14)

We assume that $c_t b_t$ denotes cost of a tree associated with block $b$, i.e., $c_t b_t = \sum_v x_{tv} c_v + \sum_w \sum_t y_{wvt} k_{wv}$, where $x_{tv}$ and $y_{wvt}$ denote the solution given by the tree. To generate the second child, we swap primary and secondary parent and apply the same procedure.

The path exchange operator also works on particular trees of individuals and its goal is to improve the quality of individual trees. The path exchange operator exchanges some of $B$ trees between two parents. All trees of the primary parent (representing blocks) are processed using the path exchange operator probability (parameter path exchange probability), i.e., either the tree is left unchanged (simply the tree is copied to the child) or the considered tree is combined with randomly selected tree of the secondary parent. In the second case, the following procedure is applied. A random node (different from the root node of the primary and the root node of the secondary tree) of the secondary tree is selected. Starting from this node, the longest possible path is selected. The selected path denoted as pathsecondary must have at least two nodes and cannot contain the root node of the primary tree. Next, starting from the root node of the considered primary tree subsequent nodes and connections are duplicated to the offspring tree. However, all nodes of the primary tree included in the pathsecondary and all descendants of these nodes (included in the primary tree) are not copied to the child tree. After that, the pathsecondary is connected to the offspring tree using the roulette method. The node to connect the path is selected by the roulette method using probability that is inversely proportional to transfer cost $c_{uv}$ between node of the offspring tree (node $w$) and the first node of pathsecondary (node $v$). Finally, all previously not processed nodes are randomly connected to the offspring tree to assure that the tree includes all nodes.
Again, to create the second offspring, we exchange primary and secondary parent and repeat the algorithm.

In the mutation process - performed according to mutation probability - we use three operators: subtree relocation selects one of individual trees and performs relocation of a random subtree within this tree; node swap chooses one of individual trees and performs swap of two, random, non-root nodes; root swap selects one of individual trees and performs swap of a random node with the tree root. Parameters percentage of relocation mutation, percentage of swap mutation, percentage of root swap mutation are used to select a particular mutation operator in a given iteration of the algorithm. Note that these three parameters must sum up to 1.

6. Results

The goal of numerical experiments was twofold: comparison of proposed heuristics against optimal results and evaluation of the computing system performance regarding various parameters including: upload and download node capacity, node processing limit, number of blocks, number of time slots and number of nodes.

6.1. Evaluation of heuristics. Heuristic algorithms PHA and PEA were implemented in C++. To find optimal results of the model (1)-(8), we applied CPLEX 11.0 (ILOG, 2009) with solver settings left at their defaults. The execution time of CPLEX solver was limited to 1 hour - consequently the CPLEX yielded three kinds of output: optimal result (execution time was lower than 1 hour), feasible results (within 1 hour the CPLEX found a solution without optimality guarantees) and no result (within 1 hour the CPLEX was not able to find any result). All experiments were made on a computer HP XW9400. Both optimal (CPLEX) solutions and proposed algorithms’ results are expressed as the unit cost values. The key factors of the solution quality are: the distance from the optimal solution and the time the result was produced. The unit values of computing and networking costs are less important in terms of solution quality and can be omitted. Thus, all results were represented as the comparison to optimal solution in order to emphasize how far they are from the best possible result. We tested 300 networks that were generated at random, since there are no existing benchmark examples. Value ranges for each parameter were set to reflect the real systems, e.g., the download capacity range was set to obtain larger values than upload capacity and thus models the most commonly used asymmetric link. Thus, the randomly obtained network structures are similar to ones met in real systems, as each parameter was separately analyzed and its randomization process was tailored to fit its specific character. Random values generation was done using C++ libraries which are believed to provide random values of uniform distribution. Main parameters (i.e., number of nodes, number of iterations and number of blocks) of the networks are described in Table 1. It should be underlined that the results presented below show some characteristic features obtained for these randomly generated systems. However, the relatively large number of tested systems and wide range of systems’ main parameters may suggest that also for other systems the major conclusions should be true. According to the performance of CPLEX, we divided the networks into three sets: small (CPLEX yields optimal results within 1 hour), medium (CPLEX provides feasible results within 1 hour) and large (CPLEX cannot find feasible results within 1 hour).

In order to compare heuristic algorithms PHA and PEA against optimal results given by CPLEX, we first performed tuning of the algorithms. To perform tuning of PHA, we selected 10 of 300 tested networks and run the algorithm changing parameter \( j_m \) from 1 to 30. All obtained results show that the average reduction of
Table 1. Parameters of tested networks.

<table>
<thead>
<tr>
<th>Type of network</th>
<th>Number of networks</th>
<th>CPLEX solution within 1 hour</th>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>53</td>
<td>Optimal</td>
<td>Number of nodes $V$</td>
<td>3-7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of time slots $T$</td>
<td>3-5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of blocks $B$</td>
<td>4-55</td>
</tr>
<tr>
<td>Medium</td>
<td>150</td>
<td>Feasible</td>
<td>Number of nodes $V$</td>
<td>5-28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of time slots $T$</td>
<td>3-18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of blocks $B$</td>
<td>6-72</td>
</tr>
<tr>
<td>Large</td>
<td>97</td>
<td>No solution</td>
<td>Number of nodes $V$</td>
<td>13-29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of time slots $T$</td>
<td>8-19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of blocks $B$</td>
<td>16-78</td>
</tr>
</tbody>
</table>

Table 2. Tuning of PEA algorithm.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
<td>100-2000</td>
</tr>
<tr>
<td>Population size</td>
<td>100-2000</td>
</tr>
<tr>
<td>Size of mating pool (percentage of population individuals selected in the tournament selection for crossover)</td>
<td>50%</td>
</tr>
<tr>
<td>Percentage of path exchange crossover</td>
<td>70%</td>
</tr>
<tr>
<td>Percentage of tree exchange crossover</td>
<td>30%</td>
</tr>
<tr>
<td>Path exchange probability</td>
<td>0.3</td>
</tr>
<tr>
<td>Tree exchange probability</td>
<td>0.3</td>
</tr>
<tr>
<td>Mutation probability</td>
<td>0.05</td>
</tr>
<tr>
<td>Percentage of relocation mutation</td>
<td>40%</td>
</tr>
<tr>
<td>Percentage of swap mutation</td>
<td>50%</td>
</tr>
<tr>
<td>Percentage of root swap mutation</td>
<td>10%</td>
</tr>
</tbody>
</table>

the OPEX cost between the case $j_m = 1$ and the case $j_m = 30$ is 17%. However, the time execution growth comparing these two cases is about 320%. But, since the execution time of PHA is relatively low (in most cases below 10 seconds), in our opinion the OPEX cost reduction is a more significant aspect. Therefore, in further experiments of the PHA algorithm we decided to set $j_m = 30$.

The next goal of experiments was to compare results of heuristics against optimal results (50 small networks). For each network, we run once deterministic PHA algorithm and 5 times the stochastic PEA algorithm. In Table 3, we report the average results taking into account all 50 networks. In the case of the PEA method, we present the mean value (fourth column), the median value (fifth column) and the minimum obtained value (sixth column). The table includes two kinds of gap to optimal results: mean value (second row), maximum obtained value (third row). Moreover, we show the number of networks, for which given algorithm yields the same result as the optimal (fourth row). Finally, we provide in the last row the average execution time of each method. Both heuristics provide results very close to optimal results yielded by CPLEX - PHA is on average 1.90% worse than optimal, the corresponding gap for the best result of PEA is only 0.40%. However, when we look at the execution time, PHA significantly outperforms other methods, since the average running time is only 20 ms. Comparison between PHA and PEA shows that the PEA algorithm is on average 1.42% better than PHA.

In the next experiment, we compared optimization methods in the context of medium networks (Table 4). In this case, CPLEX does not provide optimal results, since the execution time is limited to 1 hour. Therefore, PHA and PEA may yield results better than CPLEX. As reported in the table, PEA in the best case provides results 13.17% better (lower) than CPLEX. The corresponding value for PHA is 11.59%. The average gap between PEA and PHA is 1.40%. However, again PHA requires significantly less execution time than PEA.

Finally, we tested PHA and PEA algorithms for large networks - in this case the CPLEX cannot find any feasible result in time of 1 hour. Thus, in Table 5 only PHA and PEA algorithms are compared. Due to large execution time of PEA, for each network we run the algorithm only once. Size of tested networks and consequently size of the solution space turns out to be too large for the PEA algorithm. On average, PEA yields cost about three times larger than the value obtained for PHA, while the execution time of PEA is about 5 times larger comparing to PHA. However, simulations demonstrated us that the PEA algorithm generally works properly and improves the solution with successive iterations. Thus, we can suppose that if the running time is increased enough, the PEA algorithm should converge to a better solution.

Concluding the comparison of algorithms, we want to underline that both proposed heuristics (PHA and PEA) give solutions very close to optimal. For small and medium networks, PEA slightly outperforms PHA, however more computational time is required. For large networks, the evolutionary approach cannot cope with huge
solution space, and the PHA is much more efficient. Note that in Chmaj and Walkowiak (2010b) we examined random heuristic algorithms to solve problem (1)-(8). However, since the performance of these algorithms is much worse than PHA and PEA, we do not present these results in the paper.

6.2. P2P computing system performance. In the last part of experiments, we focused on evaluation of the considered P2P computing system regarding various parameters of the network. We made a wide range of simulations using 20 large networks (number of nodes in range 50-107, number of blocks in range 75-161, number of time slots 15).

In the following experiments, we compared performance of the P2P data delivery in contrast to a computing system using unicast flows to distributed results of computations. In Chmaj and Walkowiak (2008), we formulated an optimization model similar to [1]-[8], however the unicast transmission was applied to distribute result blocks to all nodes. Unicast - in this context - means that a particular result block $b$ can be downloaded only directly from a node $v$ that is selected to process that block ($x_{bv} = 1$).

To optimize the system with unicast flows, we developed a greedy algorithm called UHA. The general idea of UHA is similar to the PHA algorithm. The key difference between both methods is the peer selection criteria. Due to unicast flow restrictions, a node can download the result data block only from the node that has computed this certain data block. Comparing to PHA algorithm, in UHA the selection procedure becomes trivial, what significantly simplifies the algorithm. Unicast characteristic also impacts the source data allocation, which has to be tailored to fit more restrictive flow than highly flexible Peer-to-Peer flow.

The first tested parameter was the node upload capacity - we repeated simulations increasing the upload capacity of each node by 20 (up to 200). In Fig. 4 we report average (over 20 tested networks) results showing how the cost of P2P and unicast approaches changes with the increase of the upload capacity. We present average cost difference, i.e., for each particular case in terms of the upload capacity increase we calculate the percentage difference between the obtained cost and the cost of the initial case when the capacity is not increased. Since the upload capacity is increased in subsequent experiments what leads to reduction of the cost, the average cost difference is negative. Notice that the increase of the upload capacity has the largest influence in the case of the transfer cost of the P2P approach (up to 8%) - higher upload capacity enables the P2P approach to select cheaper transfers, since nodes can upload more blocks. However, the increase of the upload capacity does not reduce the processing cost of the P2P case. This follows mainly from the fact that the node processing limit is not increased. In the case of the unicast approach, both kinds of costs (processing and transfer) are reduced with the increase of the upload capacity. As mentioned above, in the case of the unicast scenario, the limit of source blocks that can be allocated to a particular node $v$ is a function of the processing limit $p_v$ and the upload capacity $u_v$. The latter constraint follows from the fact that in the unicast approach each node must have enough capacity to upload all processed blocks to remaining $(V - 1)$ nodes.

After that, we made similar experiment, i.e., we repeated simulations decreasing the upload capacity by 2 (up to 8). Fig. 5 presents average results. Again, we can watch that changing the upload capacity does not in-

---

**Table 3.** Comparison of algorithms for small networks - average distance to CPLEX results.

<table>
<thead>
<tr>
<th></th>
<th>CPLEX</th>
<th>PHA</th>
<th>PEA mean</th>
<th>PEA median</th>
<th>PEA min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean gap to optimal</td>
<td>0.00%</td>
<td>1.90%</td>
<td>0.65%</td>
<td>0.57%</td>
<td>0.40%</td>
</tr>
<tr>
<td>Maximum gap to optimal</td>
<td>0.00%</td>
<td>8.08%</td>
<td>3.76%</td>
<td>3.52%</td>
<td>1.51%</td>
</tr>
<tr>
<td>Number of optimal results</td>
<td>0.00%</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>Average execution time [s]</td>
<td>202.99</td>
<td>0.02</td>
<td>9.16</td>
<td>9.16</td>
<td>9.16</td>
</tr>
</tbody>
</table>

**Table 4.** Comparison of algorithms for medium networks - average distance to CPLEX results.

<table>
<thead>
<tr>
<th></th>
<th>CPLEX</th>
<th>PHA</th>
<th>PEA mean</th>
<th>PEA median</th>
<th>PEA min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean gap to CPLEX</td>
<td>0.00%</td>
<td>-11.59%</td>
<td>-12.15%</td>
<td>-12.31%</td>
<td>-13.17%</td>
</tr>
<tr>
<td>Maximum gap to CPLEX</td>
<td>0.00%</td>
<td>7.43%</td>
<td>3.51%</td>
<td>4.23%</td>
<td>1.52%</td>
</tr>
<tr>
<td>Minimum gap to CPLEX</td>
<td>0.00%</td>
<td>-56.55%</td>
<td>-55.58%</td>
<td>-55.58%</td>
<td>-56.02%</td>
</tr>
<tr>
<td>Average execution time [s]</td>
<td>3600.00</td>
<td>0.30</td>
<td>906.32</td>
<td>906.32</td>
<td>906.32</td>
</tr>
</tbody>
</table>

**Table 5.** Comparison of algorithms for medium networks.

<table>
<thead>
<tr>
<th></th>
<th>PHA</th>
<th>PEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean gap to PHA</td>
<td>0.00%</td>
<td>196.63%</td>
</tr>
<tr>
<td>Maximum gap to PHA</td>
<td>0.00%</td>
<td>54.28%</td>
</tr>
<tr>
<td>Minimum gap to PHA</td>
<td>0.00%</td>
<td>302.83%</td>
</tr>
<tr>
<td>Average execution time [s]</td>
<td>619</td>
<td>3359</td>
</tr>
</tbody>
</table>
Heuristic Algorithms for Optimization of Task Allocation and Result Distribution ...

In all other cases, the cost increases with the capacity decrease - the curves can be explained in the same way was in the context of Fig. 5. It should be noted that for some networks, when we decreased the original capacity by 8, the algorithms (UHA and PHA) could not find a feasible result. We also conducted experiments when the download capacity was changed (increased and decreased), however we observed that there was no impact on the cost.

The next evaluated parameter was the node processing limit related to constraint (4). The methodology was the same as in the case of previous experiments focusing on the node capacity. When the processing limit was increased for each node, there was very small impact on the cost of P2P and unicast systems. However, when we were decreasing the processing limit by 2 (up to 70), the OPEX cost was growing - Fig. 6 reports the average results. The largest influence can be observed in the case of the unicast processing cost. This is in harmony with our previous observations, i.e., when the processing limit is reduced, source blocks must be allocated to relatively more expensive nodes for processing to fulfill processing limits and capacity constraints following from construct of unicast flows. Notice that changing the processing limits has very little influence on the transfer costs.

In the following experiments, we examined the system performance when the number of blocks to be processed is changed. Since the number of blocks must exceed the number of nodes (see constraint (2)), we start experiments with $B = V$ (number of blocks equals to number of nodes) and increase the number of blocks each time adding 5 blocks to the system up to 60 new blocks. Fig. 7 shows the performance of P2P and unicast approaches.

All six presented curves referring to various costs grow nearly linear as the number of blocks increases. The largest impact is observed in the case of the P2P transfer cost. However, the relatively large growth of the P2P transfer cost is amortized by the slow increase of the P2P processing cost. These results can be explained by the fact that the P2P approach provides very flexible allocation of new blocks to cheaper nodes, as in most cases only the processing constraint limits the maximum number of blocks to be allocated to a particular node (the capacity constraint plays less significant role comparing to the unicast approach). Thus, new blocks can be allocated to cheap nodes what leads to relatively slow growth of the P2P processing cost. But at the same time, cost of P2P transfers grows more dynamically, as more expensive block transfers occur in the network. However, there is a kind of a tradeoff between both kinds of costs, as the optimization objective is the P2P OPEX cost including both elements (processing and transfer).

The next tested parameter was the number of time slots (iterations). For each tested network, we run experiments with the number of time slots in the range from 5 to 30. In Fig. 8, we report the obtained results showing how the cost changes with the increase of time slots, the reference result is the value of the cost obtained for the
case when $T = 30$. We can easily notice that for relatively small number of time slots, the yielded cost is significantly higher comparing to the reference case ($T = 30$). However, for larger values of $T (> 15)$ the average cost difference is less than 1%, i.e., the cost converges to a stable value and further increase of the iteration number does not provide substantial gain. As in the case of number of blocks (Fig. 7) also in this case (i.e., number of time slots) the P2P transfer cost is the most sensitive type of cost. The P2P transfer curve significantly differs from the unicast transfer curve. This observation confirms that the P2P approach is much more flexible comparing to the unicast case, as P2P flows more efficiently use the larger number of available time slots to reduce the OPEX cost. Moreover, in this experiment we noticed that in many cases to small number of time slots disable the algorithms to find a feasible solution.

Finally, the last parameter we focused on was the number of nodes. For each network, we increased the number of nodes by 2 up to 10 (Fig. 9). We can easily notice that the processing cost of both P2P and unicast slightly decreases when the number of nodes grows. The reason for this lies in the fact that the number of blocks is unchanged while there are more nodes to be selected for processing. Consequently, if at least one new node is cheaper than at least one existing node, the new node can be assigned with blocks and thus the processing cost is reduced. However, the transfer cost grows with the increase of nodes. This follows mainly from constraint (5) assuring that each result block must be delivered to each node, what increases the transfer cost. However, the P2P approach provides less dynamic increase of the transfer cost comparing to the unicast approach.

7. Related work

Scheduling is an important issue for efficiency of distributed computing systems. It is significantly related to Peer-to-Peer networks, which use this mechanism for data delivery. Scheduling for most P2P systems is realized in more or less centralized manner, as fully decentralized Peer-to-Peer systems are not widely used. The following work is also related with non-fully decentralized P2P structures. Scheduling for distributed systems was described by Pop (2012). The approach for multi-group multi-algorithm was presented. These ideas appear as more sophisticated scheduler, which is able to analyze the job types, their dependencies and other attributes and take them into consideration during the scheduling process. Also multiple queuing models were examined. However - typically for grid systems - the focus was put on the execution makespan and the aspect on inter-peer communication was omitted. The authors of (Vanderster et al., 2009) introduce a policy of assigning resources to grid participants. The optimization problem formulated as a variant of multichoice multidimensional knapsack problem is solved using various policies. Offline modeling of computing systems is also considered by Nabrzyski et al. (2004). Samanta et al. (2001) propose a distributed computing system for render images. In contrary to previous works, which required presence of much amount of data on each participating node - the new approach uses a division of image to primitives, which are then replicated on many nodes. Results of rendering are delivered to a central node to combine the final result image.

Many network computing systems presume that source blocks are fully independent - what means that each node can process a particular block without any knowledge about other blocks. GTapestry model described by Jin et al. (2006) enables to use relations between blocks - they may be dependent between each other. Computing nodes are divided into groups, which then compute groups of related blocks. Nodes may communicate inside groups (intra-communication) or between groups (inter-communication).

The authors of (Kim et al., 2009) use the Peer-to-Peer mechanisms over distributed computation system to implement the personalized recommendation over a customer network, which is applied for ubiquitous shopping. They operate on the Buying-net Architecture built using both hardware and software components. The proposed system offers recommendation services, which main idea is to do the personalization for each user, what should
help to take individual shopping decision. Authors propose their own model built over a proposed notation, and algorithms to solve the discussed problem. Researched systems use the P2P architecture having many peers and one central server (i.e., P2P centralized approach), which plays very minor and minimal role - most of the communication goes between peers. A peer logs into the Buying-net network (i.e., P2P network) and fetches the control information from the server, which provides the neighborhood (what is a similar idea to BitTorrent) and recommendation information. Messages regarding shopping actions are exchanged between peers. Authors pinpoint inefficiency of information dissemination in other recommendation systems and show the advantage of using Peer-to-Peer architecture in comparison with central unicast approach.

Streaming problem is also studied by Zhou et al. (2011), where authors examine the stochastic model used in several downloading strategies: Rarest First and Greedy and propose to use the mix of these approaches. Experimentation assumptions and the results are described. The P2P aspects are investigated in scope of efficient data dissemination in the group of peers. Experiments show that the proposed Mix approach is outperforming both Rarest First and Greedy approaches. We could put the idea of Mix strategy as the interesting idea worth to compare with our download (thus also upload) strategies.

Looking by this perspective, ideas presented by Miller and Wolisz (2011) are also interesting. The authors present the optimization of chunks dissemination in an overlay Peer-to-Peer network. The described problem assumes the network of peers, each having the initial set of chunks and peer link capacities. Detailed modeling and dissemination algorithms are presented together with experimentation results. Putting this work against our problem, Miller and Wolisz (2011) describes the dissemination part of our problem, without taking care of the computation phase. However, ideas presented by Miller and Wolisz (2011) may be worth to analyze as the inspiration of better solutions to our problem.

The authors of (Liu et al., 2010) present various issues related to existing distributed computing with a special focus on joint resource allocation of both computing resources and network resources. A wide range of resource allocation schemes that provide distributed computing applications with performance and reliability guarantees are described.

The idea of scheduling the flows in predefined slots, which simulate BitTorrent-like systems has been already exploited and discussed in many previous works. A time-indexed integer program defined as the Overlay Network Content Distribution is presented by Killian et al. (2005), All content to be distributed is in the form of unit-sized tokens. At the start tokens are located at one of more nodes (senders), and they are must be transferred to a different set of nodes (receivers). The distributed schedule of tokens proceeds as a sequence of timesteps. There is a capacity constraint set on each overlay arc, i.e., only a limited number of tokens can be assigned to an arc for each timestep. A token may be uploaded in a given timestep only by a node that possesses this token at the start of the timestep. The problem is proved to be NP-complete.

For more information related to P2P systems and computing systems refer to Milojicic et al. (2002), Nabrzyski et al. (2004), Shen et al. (2009), Tarkoma (2010), Travostino et al. (2006).

8. Conclusions and future work

In this paper, we studied how to optimize P2P computing systems. We described a novel architecture of a highly collaborative P2P computing system. An ILP model was formulated and discussed. Next, two heuristics: PHA and PEA were proposed. To verify proposed algorithms, extensive numerical experiments were conducted using problems instances of various size. For small networks, both heuristics yielded solutions close to optimal ones, however with significantly lower execution time (especially PHA). For medium networks PHA and PEA outperform the CPLEX solver run with 1 hour time limit again with much lower execution time. Finally, for large networks only PHA and PEA were able to find feasible results. However, the PHA provided significantly better results. Experiments proved that using the P2P approach for results delivery significantly reduces the OPEX cost comparing to the unicast approach. In our opinion, results presented in this work can be useful in designing of P2P computing systems. The algorithms presented and researched in this paper can be used in collaborative distributed systems, which strongly take the processing, networking and operating cost in general into consideration. These aspects are important not only in Internet-based applications. The non-Internet example is the network containing many mobile devices with limited packet network access. Our algorithms would, among others, decrease the amount of data transferred in such system, thus minimizing the money spent by each mobile user for packet data. The proposed approach refers to static optimization, but in our work (Chmaj and Walkowiak, 2010a) we also address the same system architecture in context of dynamic optimization.

We are aware of the fact that the formulated ILP model of the P2P computing system contains some simplifications comparing to real systems. However, even the simplified model is complicated and computationally demanding, what was shown in experiments. Moreover, in the field of the ILP optimization, in many cases constraints of the original system are reduced in order to enable formulation of ILP model that is to be solved in reasonable time. To address limitations of our approach, in
(Chmaj and Walkowiak, 2010a) we also examined the online version of the P2P computing system by using simulations. In future work, we plan to extend the ILP model formulated above to consider the following aspects: delivery of source blocks, blocks (source and result) of different size, differentiation of block starting time (blocks to be processed appear in the system in various moment of time), multiple computational projects lasting for longer time (input data is continuously generated with constant rate).

Acknowledgment

This work was supported by the Polish Ministry of Science and Higher Education (grant N N516 070435, 2008–2011).

References


Heuristic Algorithms for Optimization of Task Allocation and Result Distribution ...


Gregorz Chmaj received the M.Sc degree and Ph.D degree in computer science from the Wroclaw University of Technology, Poland, in 2005 and 2010, respectively. Currently, he is on contract for University of Nevada in Las Vegas, USA. His main areas of scientific interests are distributed processing systems and Peer-to-Peer networks, he also spent several years in computer science industry. Dr. Chmaj published several papers in journals and participated in sponsored programs.

Krzysztof Walkowiak was born in 1973. He received the Ph.D degree and the D.Sc. (habilitation) degree in computer science from the Wroclaw University of Technology, Poland, in 2000 and 2008, respectively. Currently, he is an Associate Professor at the Department of Systems and Computer Networks, Faculty of Electronics, Wroclaw University of Technology. His research interest is mainly focused on optimization of network distributed systems like P2P systems, multicasting systems, Grid systems; network survivability; optimization of connection-oriented networks (MPLS, DWDM); application of soft-optimization techniques for design of computer networks. Prof. Walkowiak was involved in many research projects related to optimization of computer networks. Moreover, he was consulting projects for the large companies including TP SA, PZU, PKO BP, Energia Pro, Ernst and Young. Prof. Walkowiak published more than 150 scientific papers. He serves as a reviewer for many international journals is actively involved in many international conferences. Prof. Walkowiak is a member of IEEE and ComSoc.

Michał Tarnawski was born in 1986. In 2005, he started his M.Sc. studies in computer science at the Faculty of Electronics, Wroclaw University of Technology, Poland. In 2010, he graduated and was awarded by Dean for the best thesis of the Faculty of Electronics in the academic year 2009/2010. His main objects of interest are computing systems, reliability and survivability of computer networks. Currently, he is employed at Opera Software International as System and Network Administrator.
Michał Kucharzak received his MSc Eng. in teleinformatics from Wroclaw University of Technology in 2008. He is currently a Ph.D. candidate at Wroclaw University of Technology and holds a position of Senior Radio Research Engineer at Wroclaw Research Center EIT+, Poland. In recent years, he has been a member of reviewer committees for many international journals, programs and technical committees for various conferences as well. His current research interests are primarily in the areas of network modeling and network optimization with special regard to overlays, simulations, design of efficient algorithms and wireless system protocols.

Received: