9-2012

Heuristic Algorithms for Optimization of Task Allocation and Result Distribution in Peer-to-Peer Computing Systems

Grzegorz Chmaj
University of Nevada, Las Vegas, chmajg@unlv.nevada.edu

Krzysztof Walkowiak
Wrocław University of Technology, krzysztof.walkowiak@pwr.wroc.pl

Michal Tarnawski
Wrocław University of Technology, mgtarnawski@gmail.com

Michal Kucharzak
Wrocław University of Technology, michal.kucharzak@pwr.wroc.pl

VERSITA

Int. J. Appl. Math. Comput. Sci., 2012, Vol. 22, No. 3, 733–748
DOI: 10.2478/v10006-012-0055-0

amcs

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 system have been gaining much attention due to a 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, contrary to grids, use personal computers and other relatively simple electronic equipment (e.g., the PlayStation console) to process sophisticated computational projects. A significant example of the P2P computing idea is the BOINC (Berkeley Open Infrastructure for Network Computing) project. To improve the performance of the computing system, we propose to use the P2P approach to distribute results of computational projects, i.e., results are 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 an 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

The Peer-to-Peer (P2P) paradigm appears to be an efficient alternative to the 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., a 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 the physical and logical structure of the 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 computing systems, also called public-resource computing systems or global computing systems.

1.1. Motivations. The chief 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 the development of various computer systems applied in many areas of human

activity. Secondly, most of the existing distributed computing systems use a centralized approach to distribute results (Anderson, 2004; Krauter *et al*., 2002; Milojicic *et al*., 2002; Nabrzyski *et al*., 2004; Shen *et al*., 2009; Travostino *et al*., 2006). Usually, the produced data are sent to a central server, which then uploads the data to requesting users. However, if a large number of users are interested in results (e.g., a collaborative research project), the server storing requested data can become overloaded or even blocked (denial of service). Therefore, in our recent work (Chmaj and Walkowiak, 2010a) we proposed an architecture of a highly collaborative P2P computing system that can efficiently distribute the results of computations. The proposed system is able to perform the computation and result distribution at the same time and assumes that the final result is combined at each node using partial results. Several types of network flows can be applied for effective data distribution. In this work, we focus mostly on P2P flows, which, according to many previous works (Shen *et al*., 2009; Tarkoma, 2010), provide the best efficiency. The final motivation for this work is to provide effective static optimization methods that can be applied to improve performance of the P2P computing systems, since, to the best of our knowledge and literature research, there are no many papers focusing on optimization of such systems. As the performance metric, we apply the OPEX cost of the system containing two elements: processing costs and transfer costs.

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 the 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 problem considered 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.

1.3. Paper overview. The remainder of the paper is organized as follows. Section 2 describes the architecture of the proposed P2P computing system. In Section 3, we formulate an ILP model of the system. Sections 4 and 5 include two heuristic algorithms (greedy and evolutionary). In Section 6, we present comprehensive numerical results showing the performance of the proposed optimization methods. In Section 7, we report related works. Finally, concluding remarks are provided in Section 8.

2. System architecture

In this section, we will briefly present the 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 distributed system considered is a P2P computing system. Note that, contrary to grids, P2P computing systems are focused on the application of personal computers and other relatively simple electronic equipment (e.g., the 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 computing nodes (e.g., personal computers) connected into one logical structure by means of the Internet. Thus, the system works in an overlay mode (Tarkoma, 2010). The underlying transport network (e.g., the Internet) provides direct connectivity between overlay nodes with the 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 existing systems the distribution of computational 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 the processing of the task, the computing node sends back the final result to the central node, which then 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 employed 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 applies a special node called the tracker. The goal of the tracker is twofold. Primary, the tracker is responsi-

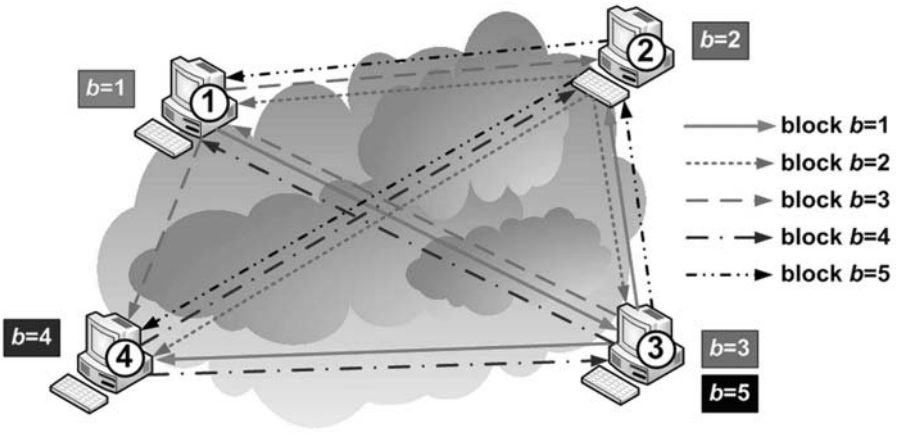

Fig. 1. Example of a distributed computing system.

ble for scheduling, i.e., it assigns individual task to computing nodes according to received requests. Next, the tracker maintains and provisions a database providing updated information related to the current state of the system, e.g., the location of calculated results. We do not use a 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 systems, refer to the works of Shen *et al*. (2009) and 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., like 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., the "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 that of our system. A key difference is that the proposed distributed system guarantees result delivery to all system participants and uses a decentralized architecture, whereas BOINC is a fully centralized system where only the managing node receives the full result. BOINC resembles unicast flow and, as we will show later in that paper, proposed P2P flow provides better results for a distributed computation system.

Figure 1 shows a simple example illustrating the main idea of the system. The system considered contains four nodes and five 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 the 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 deliver each result block to requesting nodes. The processing cost consists of expenditures related to various aspects of the IT infras-

tructure (e.g., energy, maintenance, hardware amortization, etc.). The reader interested in more information related to the economics of distributed computing systems is referred to the work of 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 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) as well as Pioro and Medhi (2004).

As mentioned above, the proposed architecture is dedicated to a highly collaborative environment, where all participants are interested in receiving 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 recent years. For instance, in 2004 Google introduced a map-reduce framework (González-Vélez and Kontagora, 2011). The inspiration comes from the map and reduce functions commonly applied in functional programming. In the map operation, the master node partitions original computing problems into smaller sub-problems and distributes them 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 the original problem. The main difference between our architecture and the map-reduce framework is the final results delivery process. In order to reduce costs of network flows, computation results are sent directly to requesting nodes, which combine the final results by themselves, 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 operations.

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 system considered includes V nodes (computing elements) indexed by $v = 1, 2, \ldots, V$. 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 compared with the transfer of result blocks. Consequently, to reduce the complexity of the ILP model, the tracker node is not included in the model. There are B computational tasks (blocks) indexed by $b = 1, 2, \ldots, B$. 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 greatest 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 the previous research on network optimization assumes a 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 T time slots (iterations), of the same size indexed by $t = 1, 2, \ldots, T$. 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 possess 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 limited processing power p_v as well as 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 the 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_{wv} denotes the cost of one result block transfer between these nodes.

To reduce the 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, the delivery of source blocks to computing nodes is not included in the time scale of the system. However, costs related to the 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

b = $1, 2, \ldots, B$, blocks (source and result)

t $= 1, 2, \ldots, T$, time slots (iterations)

v, w, s $= 1, 2, \ldots, V$, node (peer)

Constants

c_v	cost of processing of one source block in node v
k_{wv}	cost of result block transfer from node w to node v
p_v	maximum processing rate of node v
d_v	maximum download rate of node v
u_v	maximum upload rate of node v
M	large number

Variables

x_{bv}	=1, if source block b is processed (calculated) in node v; 0, otherwise (binary)
y_{bwvt}	=1, if result block b is transferred from node w to node v in iteration t; 0, otherwise (binary)

Objective

It is to find the allocation of source blocks to processing nodes and configuration of P2P transfer of result blocks to all peers minimizing the OPEX cost of the system and satisfying all constraints:

$$\min F = \sum_b \sum_v x_{bv} c_v + \sum_b \sum_w \sum_v \sum_t y_{bwvt} k_{wv}. \quad (1)$$

Constraints

(a) Since the system is collaborative and due to the fairness requirement, each peer must process at least one source block:

$$\sum_b x_{bv} \geq 1, \quad v = 1, 2, \ldots, V. \quad (2)$$

(b) Each source block must be assigned for processing to exactly one node:

$$\sum_v x_{bv} = 1, \quad b = 1, 2, \ldots, B. \quad (3)$$

(c) Each node has a processing limit p_v, therefore the number of source blocks assigned to each node cannot exceed this limit:

$$\sum_b x_{bv} \leq p_v, \quad v = 1, 2, \ldots, V. \quad (4)$$

(d) Each result block must be delivered to each peer:

$$x_{bv} + \sum_w \sum_t y_{bwvt} = 1, \quad b = 1, 2, \ldots, B, \quad v = 1, 2, \ldots, V. \quad (5)$$

Notice that this constraint is satisfied in one of two possible situations. First, the source block b is assigned to the node v for processing (i.e., $x_{bv} = 1$). Thus after processing, there is no need to transfer the result block b to the node v. Otherwise, the result block b must be delivered to the node v using the P2P approach. Therefore, the result block b must be downloaded by the node v from any other node w in one of iterations t (i.e., $\sum_w \sum_t y_{bwvt} = 1$).

(e) Upload capacity constraint:

$$\sum_b \sum_v y_{bwvt} \leq u_w, \quad w = 1, 2, \ldots, V, \quad t = 1, 2, \ldots, T. \quad (6)$$

(f) Download capacity constraint: no node can download in one time slot more blocks than its download limit:

$$\sum_b \sum_w y_{bwvt} \leq d_v, \quad v = 1, 2, \ldots, V, \quad t = 1, 2, \ldots, T. \quad (7)$$

(g) P2P flow constraint, i.e., the result block b, can be sent from a node w to a node v only if the node w keeps the source block b in the time slot t, which is equivalent to the fact that the node w either processed source block b or downloaded the result block in any iteration $i < t$:

$$\sum_v y_{bwvt} \leq M(x_{bw} + \sum_{i<t} \sum_s y_{bswi}), \quad b = 1, 2, \ldots, B, \quad w = 1, 2, \ldots, V, \quad t = 1, 2, \ldots, T. \quad (8)$$

The ILP model (1)–(8) is an NP-complete problem since it can be reduced to an MBT (Minimum Broadcast Time) problem, which is proved to be NP-complete (Jansen and Muller, 1994). The presented model can be easily extended with new elements, e.g., various sizes of result blocks, an asymmetric cost of transfer between two nodes, node join/disconnection during a system run, requirements allowing computation of some source blocks only on nodes having particular computing resources, other objective functions, e.g., throughput, processing time.

4. Greedy algorithm

In this section, we present a heuristic greedy algorithm developed to solve the optimization problem defined by (1)–(8). Note that the presented method is an improved version of an algorithm described by Chmaj and Walkowiak (2009), i.e., due to extensive simulations we modified the selection of the algorithm's parameters. The algorithm contains two methods: the first one (called PH1) yields the allocation of blocks (variables x_{bv}), the second

one (named PH2) is used to find the configuration of block transfer (variables y_{bwvt}).

The main idea of the algorithm PH1 is as follows. Detailed analysis of the model (1)–(8) shows that the model is feasible only if each node $v = 1, 2, \ldots, V$ is assigned with at least $\max(1, B - d_v T)$ source blocks. Therefore, initially we assign to each node a_v source blocks to guarantee that the problem has a feasible solution,

$$a_v = \begin{cases} B - d_v T & \text{when } d_v T < B, \\ 1 & \text{otherwise.} \end{cases} \tag{9}$$

Moreover, notice that, according to the 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 the following scoring metric:

$$e_v = mc_v + \sum_w k_{vw}, \tag{10}$$

which defines the cost of a 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 the value of the metric e_v and calculate the parameter g_v,

$$g_v = \begin{cases} 0 & \text{when } p_v = a_v, \\ \dfrac{e_{\max} - e_v}{e_v} & \text{otherwise.} \end{cases} \tag{11}$$

Finally, we allocate the remaining blocks to subsequent nodes sorted according to decreasing values of g_v. Following the 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_{bwvt}). The algorithm has one input parameter q that denotes the maximum number of block transfers in a single time slot. The idea behind the algorithm PH2 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_{wv}. Therefore, we sort all node pairs (w, v) in the ascending order of the k_{wv} metric. Next, one by one, we allocate to each analyzed node pair possibly many transfers in both directions, since the cost k_{wv} 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. The 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 limits).

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

Algorithm PHA

The algorithm uses the following input parameters: j_m is the number of algorithm's iterations, $j_m < 30$.

Step 1. Set

$$q = \left\lceil \frac{B(V-1)}{T} \right\rceil.$$

Step 2. For the following values of the parameter m: $m = 1$, $m = \lceil B/V \rceil$, $m = \lceil B/2 \rceil$ run PH1 (passing subsequent parameters m) and PH2 (passing the 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 the parameter m, for which the best feasible solution was found. Let C_0 denote the best value of the cost function.

Step 3. Calculate q_s according to the following formula:

$$g_s = \left\lceil \frac{30 - j_m}{29}(q-1) + 1 \right\rceil. \tag{12}$$

Set $j = 1$ and $q = q + q_s$.

Step 4. Run PH1 (passing the parameter m^*) and PH2 (passing the 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 $q = q + q_s$. If $q \le \sum_v u_v$ and $j \le j_m$, then go to Step 4. Otherwise, go to Step 6.

Step 6. Stop the algorithm. The result of PHA is

$$C = \min_{i=0,1,2,\ldots,j-1} (C_i).$$

In the first phase of the algorithm (Steps 1 and 2) we examine three values of the parameter m and select for further tests a value that provides the lowest value of the objective function. In the second phase (Steps 4 and 5) we focus on the parameter q. In each iteration, we increase the value of q by q_s defined in (12). We continue this process until the iteration count j does exceed the iteration limit j_m. The second stopping condition is related to the parameter q, i.e., if $q > \sum_v u_v$, the algorithm stops, since according to the construct of algorithm PH2 in such a 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 the Peer-to-peer Evolutionary Algorithm (PEA) proposed to

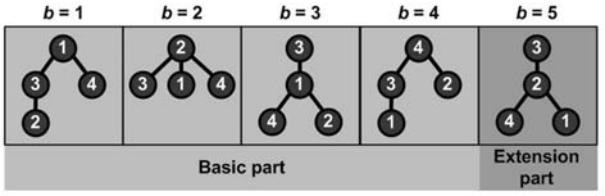

Fig. 2. Chromosome coding.

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. The root node v of the tree associated with a block b defines the node selected to process the block b, i.e., $x_{bv} = 1$. Thus, using the proposed representation we can easily obtain the values of decision variables x_{bv} and y_{bwvt}. The representation guarantees that the constraint (3) is satisfied, i.e., each source block is assigned for processing to exactly one node (the root node of the tree). Moreover, the condition (5) is also assured—each tree includes all nodes, and thus each result block is delivered to each peer. To address the 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). The remaining constraints of the problem (i.e., the constraints (4) and (6)–(8)) are satisfied by the algorithm's operators (crossover and mutation). Figure 2 the 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 four genes, the extension part includes the remaining block $b = 5$.

5.2. Algorithm overview. Figure 3 presents the block diagram of the algorithm PEA. 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 nodes are better for allocation than others. Factors that promote block allocation on a particular node v include a low processing cost (constant c_v), a low mean transfer cost to all other nodes ($k_v = \sum_w k_{vw}/(V-1)$) and a high upload limit of the node (constant u_v). According to these observations, we define below a new constant sn_v that denotes a heuristic score of node v:

$$sn_v = \frac{u_v}{c_v k_v}. \tag{13}$$

The start population is generated with the use of the roulette method—blocks are allocated on nodes with a probability proportional to the value of sn_v (13). According to our preliminary experiments, such a generation method provides significant improvements of the algorithm performance.

To generate subsequent populations, the PEA algorithm applies the tournament selection. The size of the mating pool (i.e., the number of individuals selected to perform the crossover expressed as a percentage of the *population size*) is given by an input tuning parameter called the size of the *mating pool*. Since the value of the size of the 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 a single crossover operator providing reasonable performance, we decided to apply multiple evolutionary operators. All of those operators are used randomly with probability given as the 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 the *tree exchange probability* given as a parameter. Only a tree that does not violate problem constraints is considered 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:

$$st_b = \frac{sn_v}{ct_b}. \tag{14}$$

We assume that ct_b denotes the cost of a tree associated with block b, i.e., $ct_b = \sum_v x_{bv} c_v + \sum_v \sum_w \sum_t y_{bwvt} k_{wv}$, where x_{bv} and y_{bwvt} denote the solution given by the tree. To generate the second child, we swap the primary and the 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

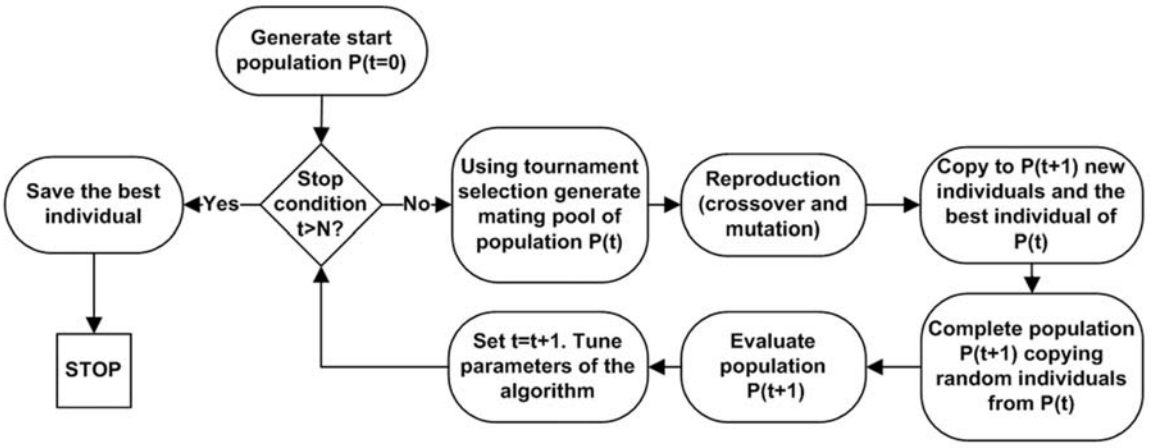

Fig. 3. Block diagram of the PEA algorithm.

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 tree considered is combined with a 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 $path_{secondary}$, must have at least two nodes and cannot contain the root node of the primary tree. Next, starting from the root node of the primary tree considered, subsequent nodes and connections are duplicated to the offspring tree. However, all nodes of the primary tree included in $path_{secondary}$ and all descendants of these nodes (included in the primary tree) are not copied to the child tree. After that, $path_{secondary}$ is connected to the offspring tree using the roulette method. The node to connect the path is selected by the roulette method using a probability that is inversely proportional to the transfer cost c_{wv} between the node of the offspring tree (node w) and the first node of $path_{secondary}$ (node v). Finally, all previously unprocessed nodes are randomly connected to the offspring tree to assure that the tree includes all nodes. Again, to create the second offspring, we exchange the primary and the secondary parent and repeat the algorithm.

In the mutation process, performed according to the *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), and *root swap* (selects one of individual trees and performs swap of a random node with the tree root). The 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 the proposed heuristics against optimal results and evaluation of the computing system performance regarding various parameters including the upload and download node capacities, the node processing limit, the number of blocks, the number of time slots and the number of nodes.

6.1. Evaluation of heuristics. PHA and PEA heuristic algorithms 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 the CPLEX solver was limited to 1 hour. Consequently, CPLEX yielded three kinds of output: an optimal result (execution time was lower than 1 hour), feasible results (within 1 hour CPLEX found a solution without optimality guarantees) and no result (within 1 hour CPLEX was not able to find any result). All experiments were conducted on an HP XW9400 computer. Both optimal (CPLEX) solutions and the 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 comparisons with the 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 real systems, e.g., the download capacity range was set to

Table 1. Parameters of tested networks.

Type of network	Number of networks	CPLEX solution within 1 hour	Parameter	Values
Small	53	Optimal	Number of nodes V	3–7
			Number of time slots T	3–5
			Number of blocks B	4–55
Medium	150	Feasible	Number of nodes V	5–28
			Number of time slots T	3–18
			Number of blocks B	6–72
Large	97	No solution	Number of nodes V	13–29
			Number of time slots T	8–19
			Number of blocks B	16–78

obtain larger values than upload capacity and thus models the most commonly used asymmetric link. Thus, the randomly obtained network structures are similar to those 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 uniformly distributed random values. The 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 presented results show some characteristic features obtained for these randomly generated systems. However, the relatively large number of tested systems and a 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 the heuristic PHA and PEA algorithms against optimal results given by CPLEX, we first performed the tuning of the algorithms. To perform tuning of PHA, we selected 10 of 300 tested networks and ran the algorithm changing the parameter j_m from 1 to 30. All obtained results show that the average reduction of the OPEX cost between the case of $j_m = 1$ and the case of $j_m = 30$ 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 $j_m = 30$.

Tuning parameters of PEA are presented in Table 2. Values of these parameters were selected according to various preliminary experiments and our experience in the field of evolutionary algorithms as well as the characteristics of the problem considered.

The next goal of experiments was to compare results of heuristics against optimal results (50 small networks). For each network, we run once the deterministic PHA algorithm and five 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: the mean value (second row) and the maximum obtained value (third row). Moreover, we show the number of networks for which a given algorithm yields the same result as the optimal one (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. A comparison between PHA and PEA shows that the PEA algorithm is on average 1.42% better than PHA.

Table 2. Tuning of the PEA algorithm.

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

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 better results than CPLEX. As reported in the table, PEA in the best case provides results

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

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

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

	CPLEX	PHA	PEA mean	PEA median	PEA min
Mean gap to CPLEX	0.00%	-11.59%	-12.15%	-12.31%	-13.17%
Maximum gap to CPLEX	0.00%	7.43%	3.51%	4.23%	1.52%
Minimum gap to CPLEX	0.00%	-56.55%	-55.58%	-55.58%	-56.02%
Average execution time [s]	3600.00	0.30	906.32	906.32	906.32

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 the PHA and PEA algorithms for large networks—in this case the CPLEX cannot find any feasible result within 1 hour. Thus, in Table 5 only the PHA and PEA algorithms are compared. Due to large exaction time of PEA, for each network we ran the algorithm only once. The size of the tested networks and consequently the 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 five times larger to PHA. However, simulations demonstrated 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 of the proposed heuristics (PHA and PEA) give solutions very close to optimal. For small and medium networks, PEA slightly outperforms PHA, although more computational time is required. For large networks, the evolutionary approach cannot cope with huge solution space, and PHA is much more efficient. Note that in our previous work (Chmaj and Walkowiak, 2010b) we examined random heuristic algorithms to solve the problem (1)–(8). However, since the performance of these algorithms is much worse than that of PHA and PEA, we do not present these results in the present paper.

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

Table 5. Comparison of algorithms for medium networks.

	PHA	PEA
Mean gap to PHA	0.00%	196.63%
Maximum gap to PHA	0.00%	54.28%
Minimum gap to PHA	0.00%	302.83%
Average execution time [s]	619	3359

In the following experiments, we compared the performance of P2P data delivery with respect to the computing system using unicast flows for distributed results of computations. In our previous work (Chmaj and Walkowiak, 2008), we formulated an optimization model similar to (1)–(8), although 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 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 that of 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. Compared to PHA algorithm, in UHA the selection procedure becomes trivial, which significantly simplifies the algorithm. Unicast characteristic also impacts source data allocation, which has to be tailored to fit a more restrictive flow than the 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 upload capacity. We present the 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

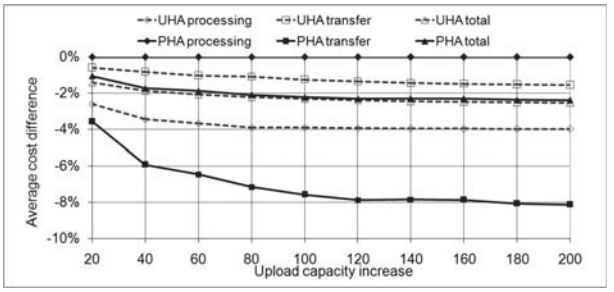

Fig. 4. Performance of P2P and unicast approaches as a function of the upload capacity increase.

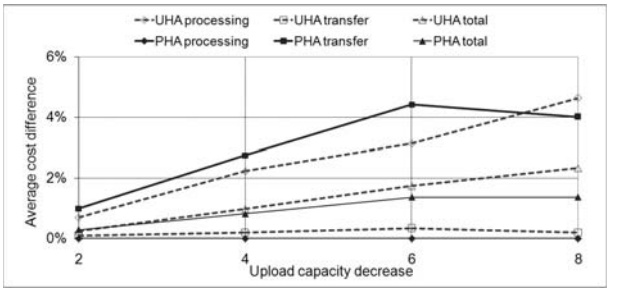

Fig. 5. Performance of P2P and unicast approaches as a function of the upload capacity decrease.

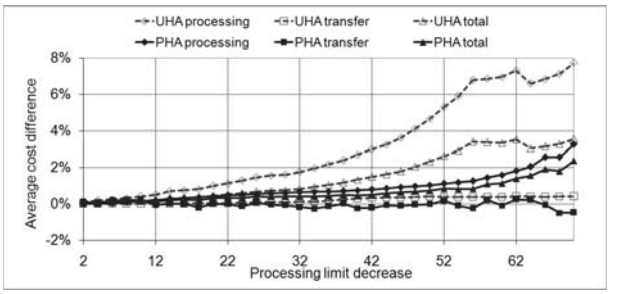

Fig. 6. Performance of P2P and unicast approaches as a function of the processing limit decrease.

initial case when the capacity is not increased. Since the upload capacity is increased in subsequent experiments, which leads to a reduction in the cost, the average cost difference is negative. Notice that the increase in the upload capacity has the greatest influence in the case of the transfer cost of the P2P approach (up to 8%)— a higher upload capacity enables the P2P approach to select cheaper transfers, since nodes can upload more blocks. However, the increase in 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 an increase in 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 the remaining $(V-1)$ nodes.

After that, we conducted a similar experiment, i.e., we repeated simulations by halving the upload capacity (up to 8). Figure 5 presents the average results. Again, we can see that changing the upload capacity does not influence the P2P processing cost. In all other cases, the cost increases with 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 a factor of 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 no impact on the cost.

The next evaluated parameter was the node processing limit related to the constraint (4). The methodology was the same as in the case of previous experiments focusing on node capacity. When the processing limit was increased for each node, there was a very low 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 greatest 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 the 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 was changed. Since the number of blocks must exceed the number of nodes (see the constraint (2)), we start experiments with $B = V$ (the number of blocks equals the number of nodes) and increase the number of blocks each time adding 5 blocks to the system, up to 60 new blocks. Figure 7 shows the performance of P2P and unicast approaches.

All the six curves presented referring to various costs grow nearly linear as the number of blocks increases. The greatest impact is observed in the case of the P2P transfer cost. However, a relatively large growth in the P2P transfer cost is compensated by a slow increase in 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 a less significant role compared with the unicast approach). Thus, new blocks can be allocated to

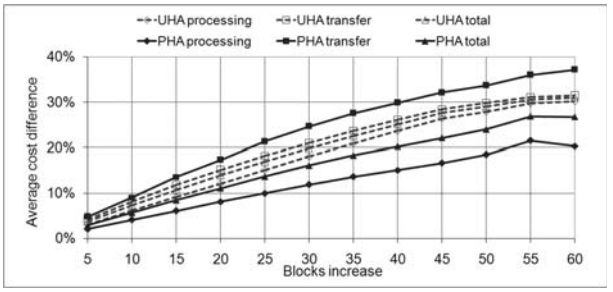

Fig. 7. Performance of P2P and unicast approaches as a function of the block increase.

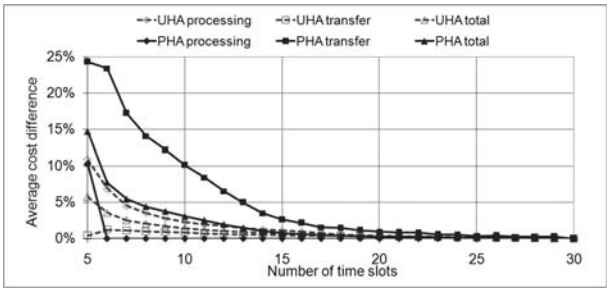

Fig. 8. Performance of P2P and unicast approaches as a function of the number of time slots.

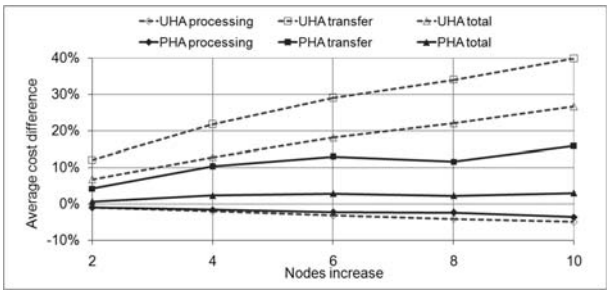

Fig. 9. Performance of P2P and unicast approaches as a function of the number of nodes.

cheap nodes, which leads to a relatively slow growth in the P2P processing cost. But, at the same time, the cost of P2P transfers grows more dynamically, as more expansive block transfers occur in the network. However, there is a kind of 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 ran 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 a relatively small number of time slots the yielded cost is significantly higher compared with 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 the number of blocks (Fig. 7), also in this case (i.e., the 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 than 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 too small a number of time slots disables the algorithms from finding 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 an increase in the number of nodes. This follows mainly from the constraint (5), assuring that each result block must be delivered to each node, which increases the transfer cost. However, the P2P approach provides less dynamic increase of the transfer cost than to the unicast approach.

7. Related work

Scheduling is an important issue for the 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 a more or less centralized manner as fully decentralized peer-to-peer systems are not widely used. The following work is also related to non-fully decentralized P2P structures. Scheduling for distributed systems was described by Pop (2012). The approach for a multi-group multi-algorithm was presented. These ideas appear as a 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 on the execution makespan and the aspect of inter-peer communication was omitted. Vanderster *et al*. (2009) introduce a policy of assigning resources to grid participants. The optimization problem formulated as a variant of the 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. Contrary to previous works, which required the presence of a large 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, which means that each node can process a particular block without any knowledge about other blocks. The GTapestry model described by Jin *et al*. (2006) allows using relations on 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).

Kim *et al*. (2009) use peer-to-peer mechanisms over a distributed computation system to implement 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, whose main idea is to do the personalization for each user, which should help to make an 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., a P2P centralized approach), which plays very minor and minimal role—most of the communication goes on 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 (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 the peer-to-peer architecture in comparison with the central unicast approach.

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

Looking from this perspective, the ideas presented by Miller and Wolisz (2011) are also interesting. The authors present the optimization of chunk dissemination in an overlay the peer-to-peer network. The described problem assumes a 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. In relation to our case, Miller and Wolisz (2011) describe the dissemination part of our problem, without taking care of the computation phase. However, the ideas presented by Miller and Wolisz (2011) may be worth analyzing as an inspiration for better solutions to our problem.

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 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 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 the works of Milojicic *et al*. (2002), Nabrzyski *et al*. (2004), Shen *et al*. (2009), Tarkoma (2010) and 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 the proposed algorithms, extensive numerical experiments were conducted using problems instances of various size. For small networks, both heuristics yielded solutions close to optimal ones, although 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, PHA provided significantly better results. Experiments proved that using the P2P approach for result delivery significantly reduces the OPEX cost compared to the unicast approach. In our opinion, results presented in

this work can be useful in designing P2P computing systems. The algorithms presented and studied 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. A non-Internet example is a network containing many mobile devices with limited packet network access. Our algorithms would, among others, decrease the amount of data transferred in such a system, thus minimizing the money spent by each mobile user on packet data. The proposed approach refers to static optimization, but in our other work (Chmaj and Walkowiak, 2010a) we also address the same system architecture in the context of dynamic optimization.

We are aware of the fact that the formulated ILP model of the P2P computing system contains some simplifications compared with real systems. However, even the simplified model is complicated and computationally demanding, which was shown in the experiments. Moreover, in the field of ILP optimization, in many cases constraints of the original system are reduced in order to enable formulation of an ILP model that is to be solved in reasonable time. To address limitations of our approach, in our previous work (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 moments 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

Anderson, D.P. (2004). BOINC: A system for public-resource computing and storage, *5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA*, pp. 4–10.

Arthur, D. and Panigrahy, R. (2006). Analyzing BitTorrent and related peer-to-peer networks, *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, SODA'06,* ACM, New York, NY, pp. 961–969, DOI: 10.1145/1109557.1109664.

BOINC (2011). BOINC poject, `http://boinc.berkeley.edu/`.

Chmaj, G. and Walkowiak, K. (2008). Data distribution in public-resource computing: Modeling and optimization, *Polish Journal of Environmental Studies* **17**(2B): 11–20.

Chmaj, G. and Walkowiak, K. (2009). Heuristic algorithm for optimization of P2P-based public-resource computing systems, *in* M. Parashar and S.K. Aggarwal (Eds.), *Proceedings of the 5th International Conference on Distributed Computing and Internet Technology, ICDCIT '08*, Springer-Verlag, Berlin/Heidelberg, pp. 180–187, DOI: 10.1007/978-3-540-89737-8_19.

Chmaj, G. and Walkowiak, K. (2010a). A P2P computing system for overlay networks, *Future Generation Computer Systems*, DOI: 10.1016/j.future.2010.11.009.

Chmaj, G. and Walkowiak, K. (2010b). Random approach to optimization of overlay public-resource computing systems, *International Journal of Electronics and Telecommunications* **56**(1): 55–62.

Christakidis, A., Efthymiopoulos, N., Fiedler, J., Dempsey, S., Koutsopoulos, K., Denazis, S. G., Tombros, S., Garvey, S. and Koufopavlou, O. G. (2011). Vital++, a new communication paradigm: Embedding P2P technology in next generation networks, *IEEE Communications Magazine* **49**(1): 84–91.

Cohen, B. (2003). Incentives build robustness in BitTorrent, `http://www.bittorrent.org/bittorrentecon.pdf`.

Couto da Silva, A.P., Leonardi, E., Mellia, M. and Meo, M. (2011). Chunk distribution in mesh-based large-scale P2P streaming systems: A fluid approach, *IEEE Transactions on Parallel and Distributed Systems* **22**(3): 451–463, DOI: 10.1109/TPDS.2010.63.

Ganesan, P. and Seshadri, M. (2005). On cooperative content distribution and the price of barter, *in* D.C. Martin (Ed.), *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, ICDCS '05*, IEEE Computer Society, Washington, DC, pp. 81–90, DOI: 10.1109/ICDCS.2005.53.

González-Vélez, H. and Kontagora, M. (2011). Performance evaluation of MapReduce using full virtualisation on a departmental cloud, *International Journal of Applied Mathematics and Computer Science* **21**(2): 275–284, DOI: 10.2478/v10006-011-0020-3.

ILOG (2009). AMPL/CPLEX software, `http://www.ilog.com/products/cplex/`.

Jansen, K. and Muller, H. (1994). The minimum broadcast time problem, *in* M. Cosnard, A. Ferreira and J. Peters (Eds.), *Parallel and Distributed Computing Theory and Practice*, Lecture Notes in Computer Science, Vol. 805, Springer-Verlag, Montreal, pp. 219–234.

Jin, H., Luo, F., Zhang, Q., Liao, X. and Zhang, H. (2006). GTapestry: A locality-aware overlay network for high performance computing, *in* P. Bellavista and C.-M. Chen (Eds.), *Proceedings of the 11th IEEE Symposium on Computers and Communications, ISCC '06*, IEEE Computer Society, Washington, DC, pp. 76–81, DOI: 10.1109/ISCC.2006.82.

Kasprzak, A. (2001). *Designing of Wide Area Networks*, Wrocław University of Technology Press, Wrocław, (in Polish).

Killian, C., Vrable, M., Snoeren, A.C., Vahdat, A. and Pasquale, J. (2005). The overlay network content distribution problem, *Technical Report CS2005-0824 UCSD*, University of California, San Diego, CA.

Kim, H.K., Kim, J.K. and Ryu, Y.U. (2009). Personalized recommendation over a customer network for ubiquitous shopping, *IEEE Transactions on Services Computing* **2**(2): 140–151, DOI: 10.1109/TSC.2009.7.

Krauter, K., Buyya, R. and Maheswaran, M. (2002). A taxonomy and survey of grid resource management systems for distributed computing, *International Journal of Software: Practice and Experience* **32**(2): 135–164, DOI: 10.1002/spe.432.

Liu, X., Qiao, C., Yu, D. and Jiang, T. (2010). Application-specific resource provisioning for wide-area distributed computing, *IEEE Network: The Magazine of Global Internetworking* **24**(4): 25–34, DOI: 10.1109/MNET.2010.5510915.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs,* 3rd Edn., Springer-Verlag, London.

Miller, K. and Wolisz, A. (2011). Transport optimization in peer-to-peer networks, *in* Y. Cotronis, M. Danelutto and G.A. Papadopoulos (Eds.), *Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP'11*, IEEE Computer Society, Washington, DC, pp. 567–573, DOI: 10.1109/PDP.2011.26.

Milojicic, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., and Z, X. (2002). Peer-to-peer computing, *Technical report*, HP Laboratories, Palo Alto, CA, HPL-2002-57.

Munidger, J. and Weber, R. (2004). Efficient file dissemination using peer-to-peer technology, *Technical report*, Statistical Laboratory Research Reports 2004-01, Cambridge.

Nabrzyski, J., Schopf, J.M. and Weglarz, J. (Eds.) (2004). *Grid Resource Management: State of the Art and Future Trends*, Kluwer Academic Publishers, Norwell, MA.

Pioro, M. and Medhi, D. (2004). *Routing, Flow, and Capacity Design in Communication and Computer Networks*, Morgan Kaufman Publishers, San Francisco, CA.

Pop, F. (2012). Heuristics analysis for distributed scheduling using MONARC simulation tool, *ICMS 2012: International Conference on Modeling and Simulation, Zurich, Switzerland*, pp. 157–163.

Przewozniczek, M., Walkowiak, K. and Wozniak, M. (2011). Optimizing distributed computing systems for k-nearest neighbors classifiers: Evolutionary approach, *Logic Journal of IGPL* **19**(2): 357–372, DOI: 10.1093/jigpal/jzq034.

Ramzan, N., Quacchio, E., Zgaljic, T., Asioli, S., Celetto, L., Izquierdo, E. and Rovati, F. (2011). Peer-to-peer streaming of scalable video in future Internet applications, *IEEE Communications Magazine* **49**(3): 128–135, DOI: 10.1109/MCOM.2011.5723810.

Samanta, R., Funkhouser, T. and Li, K. (2001). Parallel rendering with k-way replication, *in* S.N. Spencer (Ed.), *Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, PVG'01*, IEEE Press, Piscataway, NJ, pp. 75–84.

Shen, X., Yu, H., Buford, J. and Akon, M. (Eds.) (2009). *Handbook of Peer-to-Peer Networking,* 1st Edn., Springer Publishing Company, New York, NY.

Stutzbach, D., Zappala, D. and Rejaie, R. (2004). Swarming: Scalable content delivery for the masses, *Technical Report CIS-TR-2004-1*, University of Oregon, Eugene, OR.

Suri, N., Benincasa, G., Tortonesi, M., Stefanelli, C., Kovach, J., Winkler, R., Kohler, R., Hanna, J., Pochet, L. and Watson, S. (2010). Peer-to-peer communications for tactical environments: Observations, requirements, and experiences, *IEEE Communications Magazine* **48**(10): 60–69.

Tarkoma, S. (2010). *Overlay Networks: Toward Information Networking,* 1st Edn., Auerbach Publications, Boston, MA.

Terzo, O., Mossucca, L., Cucca, M. and Notarpietro, R. (2011). Data intensive scientific analysis with grid computing, *International Journal of Applied Mathematics and Computer Science* **21**(2): 219–228, DOI: 10.2478/v10006-011-0016-z.

Travostino, F., Travostino, F. and Karmous-Edwards, G. (Eds.) (2006). *Grid Networks Enabling Grids with Advanced Communication Technology*, Wiley, Chichester.

Vanderster, D.C., Dimopoulos, N.J., Parra-Hernandez, R. and Sobie, R.J. (2009). Resource allocation on computational grids using a utility model and the knapsack problem, *Future Generation Computer Systems* **25**(1): 35–50, DOI: 10.1016/j.future.2008.07.006.

Wozniak, M. (2009). Evolutionary approach to produce classifier ensemble based on weighted voting, *in* A. Abraham, A. Carvalho, F. Herrera and V. Pai (Eds.), *Nature and Biologically Inspired Computing*, IEEE, Coimbatore, pp. 648–653.

Yang, X. and de Veciana, G. (2004). Service capacity of peer to peer networks, *INFOCOM 2004. 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China*, Vol. 4, pp. 2242–2252.

Zhou, Y., Chiu, D.-M. and Lui, J.C.S. (2011). A simple model for chunk-scheduling strategies in P2P streaming, *IEEE/ACM Transactions on Networking* **19**(1): 42–54, DOI: 10.1109/TNET.2010.2065237.

Grzegorz Chmaj received the M.Sc. and Ph.D. degrees in computer science from the Wrocław University of Technology, Poland, in 2005 and 2010, respectively. Currently, he is under contract with the University of Nevada in Las Vegas, USA. His main areas of scientific interests are distributed processing systems and peer-to-peer networks, but he has also spent several years in computer science industry. Doctor Chmaj has 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 Wrocław University of Technology, Poland, in 2000 and 2008, respectively. Currently, he is an associate professor in the Department of Systems and Computer Networks, Faculty of Electronics, Wrocław 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. Professor Walkowiak has been involved in many research projects related to optimization of computer networks. Moreover, he has consulted projects for large Polish companies including TP SA, PZU, PKO BP, Energia Pro, or Ernst and Young. Professor Walkowiak has published more than 150 scientific papers.

Michał Tarnawski was born in 1986. In 2005, he started his M.Sc. studies in computer science at the Faculty of Electronics, Wrocław University of Technology, Poland. In 2010, he graduated and was awarded for the best thesis at 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 a system and network administrator.

Michał Kucharzak received his M.Sc. degree in teleinformatics from the Wrocław University of Technology in 2008. He is currently a Ph.D. candidate at the same university and holds the position of a senior radio research engineer at the Wrocław Research Center EIT+, Poland. In recent years, he has been a member of reviewing committees for many international journals, as well as program and technical committees for various conferences. 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: 8 September 2011
Revised: 28 March 2012