HEURISTIC ALGORITHM FOR OPTIMIZATION OF PEER-TO-PEER COMPUTING SYSTEMS

Abstract
Recently, Peer-to-Peer computing systems have been gaining much attention due to the growing requirement for data processing. We focus on the optimization problem related to computations and data transmission in overlay-based P2P computing systems with the system cost objective. An effective heuristic algorithm is proposed and compared against optimal results.

Key words
P2P, computing systems, overlay, optimization.

1. Introduction
There is a growing need for large computational power to answer various challenges related to various aspects of our lives including: financial modeling, medical data analysis, experimental data acquisition, earthquake simulation, and climate/weather modeling, astrophysics and many others [4], [10], [15]. Therefore, computing systems including Peer-to-Peer (P2P) systems, Grids, Web services, cloud computing, are becoming popular in both academia and industry.

We focus on P2P computing system also known as global computing or public-resource computing, which is mainly focused on the application of personal computers and other relatively simple electronic equipment instead of supercomputers and clusters [8]. An significant case of public-resource computing project is SETI@home started in the 1999 [1].

In this work we formulate an optimization problem related to P2P computing system. We assume that the computing system uses overlay network. The goal is to minimize the cost of the system including both: computation cost and transmission cost. The Integer Programming (IP) formulation of the optimization problem is used to obtain optimal results. Moreover, a new heuristic algorithm is proposed, which – according to results of numerical experiments – provides results close to optimal.

The rest of the paper is organized as follows. In the next section we formulate the P2P computing system optimization problem. Section 3 includes a description of the heuristic algorithm. In section 4 we present results of numerical experiment. Finally, in the last section some concluding remarks are provided.

2. Optimization model
In this section we present the Integer Programming model of P2P computing systems. Assumptions of the follow mainly from the construct of BOINC system [1] and recommendations of earlier authors included in [6-10], [11-18]. Since our focus is on data transmission we do not address in detail several vital problems of network computing systems such as: management, security, diverse resources. However, due to the layered architecture of both: computer networks (e.g. ISO/OSI, TCP/IP) and computing systems (e.g. Globus Toolkit) our model can be used in many scenarios together with various protocols and technologies related to computer networks and computing systems.

Peers participating in the public-resource computing system (PCs or other computers) are represented as vertices (network nodes) denoted using index \( v = 1,2,\ldots,V \). Each vertex \( v \) is connected to the overlay network using an access links with limited download rate \( d_v \) and upload rate \( u_v \). Each vertex \( v \) has a limited processing power \( p_v \) (e.g. CPUs, FLOPS) that denotes how many uniform jobs can be calculated on \( v \) in a particular time.

The computational project to be calculated in the public-resource computing system is divided into uniform jobs having the same processing power requirement (e.g. CPUs, FLOPS). With each job we associate the notion of a block, i.e. data that is generated due to the processing of the job. Every block is of the same size and is denoted as \( b = 1,2,\ldots,B \). For simplicity, in the remainder of the paper we use the term block in two senses: computational job and data block.

One of the most difficult problems encountered in modeling of P2P systems is the time scale. As in [6], [8], [16-18] we propose to divide the time scale into time slots of the same length, that can be interpreted also as subsequent iterations of the systems. We use index \( t = 1,2,\ldots,T \) to denote subsequent time slots. In each iteration \( t \), vertices may transfer blocks between them.

Each block (job) must be assigned to exactly one vertex for processing. We use the decision binary variable \( x_{bv} \) to denote the assignment (scheduling) of block \( b \) to vertex \( v \) for processing. The second decision variable – \( y_{bvw} \) – is associated with transmitting of blocks transfer. The decision binary variable \( y_{bvw} \) equals 1 if block \( b \) is transferred to node \( v \) from node \( w \) in iteration \( t \), 0 otherwise. In this work we assume that a result block \( b \) can be downloaded only from node \( v \) that computed block \( b \), i.e. \( x_{bv} = 1 \). This means that the unicast transmission is used to deliver the result blocks. In our previous work we have considered also other transmission concepts: P2P transfers, anycast and multicast [16-17]. Note that both variables are coupled – scheduling of blocks influences the transfer process.

The computational project is collaborative – each peer of the computing system (represented by the vertex) wants to receive the whole output of processing. For the
sake of fairness of the system, we assume that each vertex participating in the system must be assigned with at least one block for processing.

In our approach we do not model transmitting of the input data for processing. This data is delivered prior to initiation of the computing system. In other words, the time scale of our system begins when all source blocks are calculated on vertices. This assumption is motivated by the fact that usually source data is offloaded from one network site. Let’s assume that the size of input and output data is the same. To transfer input blocks we need at most \( B \) (number of all blocks) transfers in the overlay network, because each block must be delivered to exactly one vertex. To transmit the output data to all participants we need \( B(V-1) \) transfers, where \( V \) is the number of all vertices. From this simple example we can see that if input and output data is of comparable size, much more network traffic is issued in the output data delivery. Cost of source data delivery is included in the cost of processing block \( b \) on node \( v \). However, models presented below can be easily modified to incorporate also source data delivery. For simplicity, we make an assumption that the download and upload rates of vertices are expressed in blocks per time slot.

The objective is to minimize the system cost that includes: processing cost of block \( b \) in vertex \( v \) denoted as \( c_v \), and the cost of transfer from source vertex \( w \) to destination vertex \( v \) denoted as \( k_{vw} \). The processing cost can include all aspects of IT infrastructure (energy, maintaining, hardware amortization etc.). Various issues of grid economics can be found [10]. The second part of the criterion function is associated with the transmission cost \( k_{vw} \) between vertices \( w \) and \( v \). Constant \( k_{vw} \) can be interpreted in several ways, e.g. economical, network delay, number of hops, RTT. For a good survey on participating costs in a P2P network refer to [3].

Now, we present the optimization problem using the notation proposed in [11].

\[
\begin{align*}
\text{indices} & \quad b = 1,2,...,B \quad \text{(blocks (jobs) to be processed)} \quad t = 1,2,...,T \quad \text{(time slots (iterations))} \quad v,w = 1,2,...,V \quad \text{(peers, overlay network nodes)} \\
\text{constants} & \quad c_v \quad \text{(cost of block processing (calculation) in node } v) \\
& \quad k_{vw} \quad \text{(cost of block transfer from node } w \text{ to node } v) \\
& \quad p_v \quad \text{(maximum processing rate of node } v) \\
& \quad d_v \quad \text{(maximum download rate of node } v) \\
& \quad u_v \quad \text{(maximum upload rate of node } v) \\
& \quad M \quad \text{(large number)} \\
\text{variables} & \quad x_{bv} \quad 1 \text{ if block with index } b \text{ is processed in node } v; 0 \text{ otherwise (binary)} \\
& \quad y_{bvw} \quad 1 \text{ if block } b \text{ is transferred to node } v \text{ from } w \text{ in iteration } t; 0 \text{ otherwise (binary)} \\
\text{objective} & \quad \min F = \sum_b \sum_v x_{bv} c_v + \sum_v \sum_w \sum_t y_{bvw} k_{vw} \\
\text{subject to} & \quad \sum_b x_{bv} \geq 1 \quad v = 1,2,...,V \quad (2) \\
& \quad \sum_v x_{bv} = 1 \quad b = 1,2,...,V \quad (3) \\
& \quad \sum_b x_{bv} \leq p_v \quad v = 1,2,...,V \quad (4) \\
& \quad x_{bv} + \sum_w \sum_t y_{bvw} = 1 \quad b = 1,2,...,B \quad v = 1,2,...,V \quad (5) \\
& \quad \sum_v \sum_w y_{bvw} \leq u_w \quad w = 1,2,...,V \quad t = 1,2,...,T \quad (6) \\
& \quad \sum_v \sum_w y_{bvw} \leq d_v \quad v = 1,2,...,V \quad t = 1,2,...,T \quad (7) \\
& \quad \sum_v \sum_w y_{bvw} \leq M x_{bv} \quad b = 1,2,...,B \quad w = 1,2,...,V \quad (8) \\
\end{align*}
\]

Constraint (2) guarantees that each vertex must process at least one block. (3) assures that each block is assigned to only one vertex. Constraint (4) is the limit on processing power. To meet the requirement that each vertex (peer) must receive all blocks we introduce condition (5). Notice that block \( b \) can be assigned to node \( v \) for processing \( (x_{bv} = 1) \) or block \( b \) is transferred to node \( v \) in one of iterations \( (y_{bvw} = 1) \). Constraints (6) and (7) are upload and download capacity constraints. Condition (8) assures that vertex \( w \) cannot upload block \( b \) if the block was not computed in \( w \). Note if we want to use other transmissions concepts (P2P, anycast or multicast) the constraint (8) must be modified [16-17]. For instance, in the case of P2P flows it should be as follows

\[
\sum_v \sum_w y_{bvw} \leq M x_{bv} + \sum_{v < w} \sum_t y_{bvw} \quad b = 1,2,...,B \quad w = 1,2,...,V \quad t = 1,2,...,T \quad (9)
\]

Problem (1-8) is an NP-complete problem, because it can be reduced to the knapsack problem.

3. Algorithm

In this section we propose a heuristic algorithm called UHUA designed to solve problem (1)-(8). Algorithm UHUA (Unicast Heuristic Algorithm) consists of two sub-algorithms UH1 and UH2 which we will describe in the following. Algorithm UH1 assigns source blocks to nodes, and algorithm UH2 performs distribution of result blocks among all nodes.

UH1 algorithm starts with assignment of minimal blocks’ number to each node. Specific unicast model constraints enforce the fact, that each node must have at least \( a_v \) blocks assigned – otherwise it is unable to satisfy constraint (5).

\[
a_v = \begin{cases} 
B - d_v \cdot T & \text{if } B - d_v \cdot T > 0 \\
1 & \text{otherwise}
\end{cases}
\]

(9)

If all blocks are assigned \( \sum_v a_v = B \) – UH1 algorithm finishes its work, otherwise it continues with further assignment. Nodes for which additional blocks will be assigned are selected based on scoring mechanism. For each node, score value \( e_v \) is computed using following formula.

\[
e_v = c_v + \sum_w k_{vw}
\]

(10)
4. Results

To verify effectiveness of the UHA algorithm we built a dedicated simulation system in C++. It is composed of two modules: optimal solver, heuristic algorithms module. The optimal solver module applies CPLEX 11.0 library [5]. The execution time of CPLEX was limited to 3600 seconds. Consequently, it provides optimal solution for small networks and feasible (more or less close to optimal result) solutions for larger structures. Problem instances used in numerical experiments were generated by our own network generator.

The main objective of experiments is to compare algorithm UHA against optimal and feasible results yielded by CPLEX. We consider various scenarios in terms of the proportion between processing cost and transfer cost. In Table 1 we report the comparison between UHA and CPLEX. We can see that for smaller networks (134 cases) in all cases UHA provides the optimal result. For larger networks (58 cases), the CPLEX execution time was 3600 seconds and the solution yielded by CPLEX has only the feasible status (no guarantee to be optimal). In 47 of 58 networks the UHA algorithm provided the same result as CPLEX. In 11 of 58 cases UHA outperformed CPLEX and the average gap was -0.05%. It should be underlined, that in case of larger networks, the execution time of UHA was less than 1 second, while CPLEX was run for 3600 seconds.

Table 1

<table>
<thead>
<tr>
<th>Number of networks</th>
<th>Number of nodes</th>
<th>Number of blocks</th>
<th>Number of iterations</th>
<th>CPLEX status</th>
<th>UHA vs. CPLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>134</td>
<td>3-15</td>
<td>3-16</td>
<td>2-8</td>
<td>optimal</td>
<td>0.00%</td>
</tr>
<tr>
<td>58</td>
<td>12-18</td>
<td>13-21</td>
<td>6-9</td>
<td>feasible</td>
<td>-0.05%</td>
</tr>
</tbody>
</table>

On Fig. 1 we present the performance of UHA algorithm for various networks. Fig. 2 shows parameters of tested networks.
On Fig. 3 we report the P2P computing system cost (yielded by UHA algorithm) as a function of the number of blocks. We can observer that the system cost grows proportionally to the number of blocks.

![Fig.3. System cost as a function of block number](image)

5. Concluding remarks

In this paper we have addressed the optimization problem related to P2P computing systems. A new heuristic algorithm was proposed. The numerical evaluation has shown that the proposed algorithm for smaller networks in all cases provides the optimal results. For larger networks it can outperform the CPLEX solver with 1 hour time limit. The presented algorithm can be applied for offline optimization of P2P computing systems. Obtained results can be used as benchmark that enables to evaluate the effectiveness of distributed systems working in online manner. In the future work we want to develop a simulation system to verify various scheduling strategies used in distributed environment.

6. Acknowledgment

This work is supported by The Polish Ministry of Science and Higher Education under the grant which is being realized in years 2008-2011.

References


GRZEGORZ CHMAJ
grzegorz@chmaj.net
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

KRZYSZTOF WALKOWIAK
krzysztof.walkowiak@pwr.wroc.pl
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław