1-2013

A P2P Computing System for Overlay Networks

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

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

A P2P computing system for overlay networks

Grzegorz Chmaj[a], Krzysztof Walkowiak[b]

[a] Department of Systems and Computer Networks, Wroclaw University of Technology
Wybrzeze Wyspianskiego 27. 50-370 Wroclaw, Poland

[b] Department of Systems and Computer Networks, Wroclaw University of Technology
Wybrzeze Wyspianskiego 27. 50-370 Wroclaw, Poland
krzysztof.walkowiak@pwr.wroc.pl (corresponding author)

Abstract

A distributed computing system is able to perform data computation and distribution of results at the same time. Computing systems consist of many machines, which jointly constitute a large computation power that would not be available on a single machine. The input task is divided into blocks, which are then sent to system participants, which offer their resources in order to perform calculations. The computing of a block produces a partial result, which is sent back by the participant to the task manager (usually one central node) where all partial results are combined into the final result. In the case when system participants want to get the final result, the central node may become overloaded, especially if many nodes will request the result at the same time. In this paper we propose a novel distributed computation system, which does not use the central node as the source of the final result, but assumes that partial results are sent between system participants. This way we avoid the overload of the central node, as well as the network congestion. There are two types of distributed computing systems: grids and public computation systems (called also 'Peer-to-Peer computing systems'). In this work we focus on the latter case. Consequently, we assume that the computing system works on the top of an overlay network. We present a complete description of the P2P computing system, considering both computations and result distribution. To verify the proposed architecture we use our own simulator. The obtained results show the system performance expressed by the operation cost for various types of network flows: unicast, anycast and Peer-to-Peer. Moreover, the simulations prove that our computing system provides about 66% lower cost comparing to a centralized computing system.

Keywords: computing systems, P2P, simulation, overlay networks

1. Introduction

Distributed computing systems play a very significant role in today's academic and business world. This kind of systems consists of many machines connected to one computational grid, which is considered as one virtual machine with a large computation power. Distributed computing systems are applied to compute tasks requiring huge

computation power which is not available on a single machine (even on a super-computer). They are mainly divided into two categories: grid computing systems and Peer-to-Peer computing systems. Grid systems are constituted by organizations and institutions and contain a small number (usually up to hundreds) of machines [1], [2] connected using network links of high efficiency. Grids may share many kinds of resources: computing power, disk space, data, sensors, etc [3]. Resources are centrally managed using systems such as RMS (Resource Management System) and cover the following aspects: customizability, extensibility, scalability, etc. [4]. Scheduling is an important element of the grid that has a large influence in the system efficiency [2], [5], [6]. It should include such issues as: resource discovery, information gathering and task execution, concurrently with authorization, application management and monitoring [5]. Many papers assume simplifications of the scheduling model, correspondingly in this paper we focus on one aspect of scheduling, i.e., assignment of computational tasks to computing nodes.

Constituting the computing grid system is a sophisticated task regarding both technical and financial aspect. Therefore, other distributed computing systems – called Peer-to-Peer (P2P) computing systems – have emerged in recent years [7]. These systems are built using many private machines, which are most often home PC or Macintosh computers or even gaming consoles. The user installs computing software on her/his machine and registers into a selected computing project. Then, she/he receives data chunks to compute and send the results back to the central node, where partial results received from users are combined into the final result. Network connections used in P2P computing systems are regular home access links: such as DSL or cable. This approach is much simpler than grids, since the only requirement is to provide suitable software and to manage tasks and results (in the case of grid systems, also physical machines must be maintained). P2P computing allows for unreliability of participants – they may freely join and leave the computational grid, which is not used in grid systems. The most popular P2P computing project is SETI@home (started in 1999), which aims at looking for an extra terrestrial intelligence [8]. It is based on a BOINC architecture [7], [9]. Projects based on the BOINC aggregate almost 2 million users all over the world with over 5 million hosts having 5 TeraFLOPS of power (April 2010). Seti@home is the largest BOINC P2P computing project (over 1 million of users), other popular projects are: Einstein@home (250 thousand users – search for pulsar stars) and Climate Prediction (224 thousand users – climate change prediction). There are also other Peer-to-Peer computing frameworks, including systems dedicated to compute one project, e.g. [10], [11].

Grid systems are mostly centrally managed, what means that there is one central node, which takes care of task preparation, scheduling and managing of results. P2P computing systems may also use this model, but as home users contribute with their resources, they may also want to participate in the results. This entails the problem of distributing the complete result to each of the participant. In the case when the result is combined at one central node, huge number of participants interested in the results and requesting it from one central machine may cause the server overload or even denial of service. For instance, the authors of *Electric Sheep* project [12] propose a distributed computing system, which renders artificial forms of life – and allows participants to get complete animation. The animation is rendered by participants, but combined into the final result at central node. The authors underline that their system struggles with the problem of downloading the final animation from the central node and plan to use BitTorrent [13] protocol to solve this issue.

In this paper we propose a new idea of a distributed computing system. Our system is able to perform computation and result distribution at the same time. The main novelty is that the system is not centrally managed – partial results are not sent back to the central node, but transferred between nodes directly. Similarly to the BitTorrent protocol [13], in our system we use a special node called tracker. The objective of the tracker in the computing system is twofold. First, the tracker performs the scheduling, i.e., the node assigns individual task to computing nodes according to received requests. Second, the tracker maintains and offers the current database including information on location of already calculated results.

Distributed computing systems are often modeled with a static approach, which assumes the creation of a static optimization model (including decision variables, constraints and objective function) [14], [15]. Other popular approach to research on distributed systems is the simulation – which is based on dedicated software and aims to act as close to a real (modeled) system as possible.

The considered P2P computing system works in an overlay mode and uses the Internet as a transport layer. The overlay approach assumes that the network includes two layers: upper application layer and lower transport layer [16], [17]. The transport layer provides direct connectivity between overlay nodes. Moreover, some Quality of Service guarantees can be assured by the transport layer. Each node (participant of the computing system) is connected to the overlay network by an access link with specified download and upload bandwidth expressed in bps.

The motivation for this paper is to propose a novel approach to distributed computing systems, as today’s systems do not provide effective mechanisms to deliver results of

computations to all participants. Main contributions of this paper are as follows. (1) A novel architecture for a P2P computing system considering both computations and results dissemination. (2) Decision strategies developed for computing nodes participating in the system. (3) A simulator of the proposed distributed computing system implementing various types of network flows (unicast, anycast and P2P). (4) Simulation results showing the influence of network flow and proposed policies.

The remainder of the paper is as follows. In Section 2 we introduce the P2P computing system in detail. Section 3 includes the description of the simulator developed to examine the proposed system. In Section 4 we show results of the experiments. Section 5 includes the related work. Finally, the last section concludes this work.

2. The proposed Peer-to-Peer computing system

In this section we present the architecture of a new P2P computing system. The main objective of the system is to minimize the OPEX cost of the system compromising both: computing and transfer costs. The former element refers to operating costs related to computation (e.g. energy, maintenance). The latter cost is the delivery cost in the overlay network usually defined for each pair of nodes (e.g. lease cost of the access links). We assume that the system is collaborative and all participants want to get the whole result. However, our architecture could be also easily used to model the situation when only some of participants download the result.

The system consists of many machines connected into one logical structure. It takes sophisticated computational task as the input, which is then computed by participants. Idea of distributed computation is used like in many systems such as grids [1], [2] cloud computing [18] and P2P computation systems [10], [7]. As delivery of all results to each participant introduces significant network traffic, it is essential to provide effective distribution algorithms. Like in most of distributed computation systems (e.g., Seti@Home and many others based on BOINC [7] framework) the input data is divided into uniform blocks (we call them *source blocks*), which are sent to system participants in order to be computed. BOINC systems assume that the result of computation (*result block*) is sent back to the central node, which collects all results for further processing and analysis. In contrary, our research considers situation, when all system participants are interested in the final results and results are not sent back to the central node, but they are disseminated over the network to all requesting users. Examples of this problem include: distributed images rendering or 3d movies distribution rendering, where all participating users want to see the result of rendering.

In the case when the central node is used to combine partial results into the final result, download performed by many users cause high load and congestion problems at the central node. We investigate how to bypass the central node and use the advantages of various flows, to optimize result distribution.

Let us now describe the details of our system architecture. It contains two types of elements:

- *nodes* – regular machines that do computation and exchange results between each other
- *tracker* – a central element, which assigns source blocks to nodes. It is also used as a database about result locations, what is similar to the idea of tracker node in BitTorrent protocol. Each node has its own locations base, which is periodically updated with data obtained from the tracker.

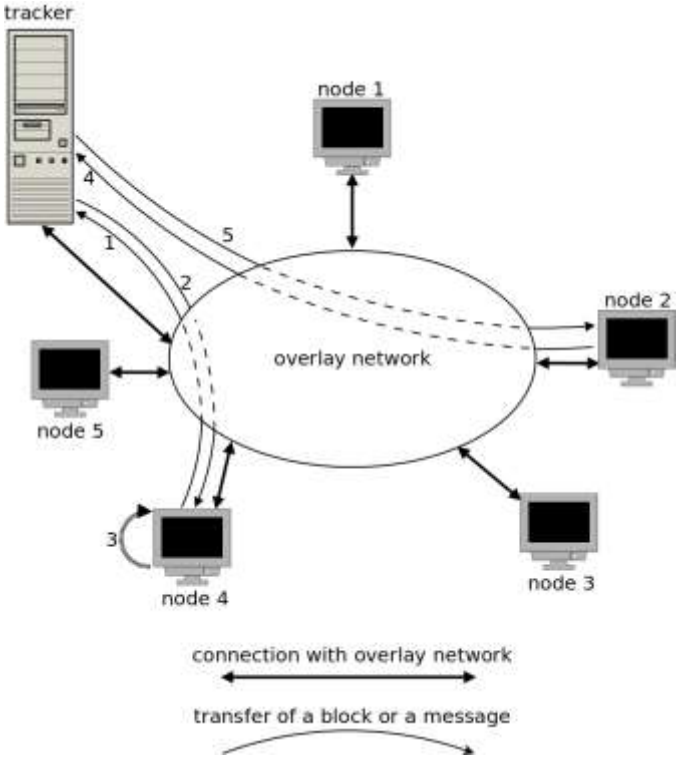

Fig. 1. Operations in a simulations system

A node requests the source block from the tracker when it has free computation resources available (operation 1 in Fig. 1). The tracker responds with the source block if available (operation 2) or signals that no more source blocks are available for computation, i.e., all blocks included in the current computing project have been assigned to nodes for computation. When the node receives such information, it stops requesting new blocks from

the tracker. The node has to compute at least one source block (operation 3), to become the participant of the project. The tracker stores information about project participants and does not provide information about results locations for nodes, which are not present on the participants list. This way the system protects itself against unfairness – every node has to contribute to the system in order to obtain final output results. A node that wants to get the result, which was computed by other node, sends a location request to the tracker (operation 4). Then, the tracker responds with known locations (operation 5). The node selects one of them according to the selection policy (decision strategy). To make the tracker locations' list complete, the node sends the update to the tracker every time it acquires a new block available to send. This happens in two cases: the node has finished computing the source block or the node has finished downloading the result block from the other node.

All elements of the system (nodes and tracker) are connected through the overlay network, which is the Internet in our case. This way we consider a network as one unified structure, that provides a direct connection between every two elements connected to this network. We do not consider how such connection (in our case: between two IP addresses) is established, as this happens at the lower layer (routing). Thus, a full mesh of connections is assumes – each peer can directly connect to any other peer.

Each node (denoted by index $v = 1,2,\ldots,V$) is characterized by several parameters describing its ability and effectiveness in joint cooperation. Power of CPUs of each node is expressed by the processing power factor denoted as p_v. This factor describes the ability of a node for computing source blocks. Also other type of hardware components may be included into p_v, as this is often to use various type of hardware for P2P computations (e.g., graphical processing units or whole Sony Playstation devices may be used for regular data computation). Each node is connected to an overlay network through a link, it is often DSL, cable or even wireless GSM/3G access type. Thus, we define upload speed u_v and download speed d_v for each node participating in the P2P computing system. For the tracker we do not define neither the computing power (as it does not compute blocks) nor the link speed. We motivate this simplification by the fact that the tracker element in popular systems like BitTorrent or the centralized element in BOINC system is usually a large machine with high speed network link. Nodes operate simultaneously and autonomously. Each node may perform many different actions at the same time: send a request to the tracker, send a request to other node, compute source blocks, send and receive result blocks from other nodes.

Node's resources – upload speed, download speed and processing power – are managed using *channels*. In each type of the resource, there is one channel dedicated to the

communication with the tracker. Since the overhead of processing and transmission of signaling messages between the peer and the tracker is relatively small compared to other operations of the node, only small amount of each resource (link speed and processing power) is assigned to this task. The rest of each resource is divided proportionally into a fixed number of channels (e.g., 4). The processing channel is used for computation of blocks. Consequently, in a given moment of time, the peer can process a limited number of blocks. The upload and download channels are used to send both data (blocks) and signaling messages to other peers. Note, that the channel is occupied until the operation is finished. This way the node may handle many independent tasks at the same time. The idea of channels follows from the BitTorrent system, in which each node can have a limited number of active peers and the default value of this parameter is 4 [13].

Nodes and the tracker may interact between each other by sending signaling messages. We distinguish the following kinds of messages in the system:

- *source block request* – is sent to the tracker from a node, which wants to obtain a new source block for computing;
- *tracker update* – is sent to the tracker from a node that starts to posses a new result block available for download, this message updates the tracker's location list
- *block location request* – is sent to the tracker from a node, which wants to receive a current list of nodes that posses result blocks;
- *download request* – is sent from a node that misses a result block to another node, which has the requested result block available for download. It must be confirmed by the *download acknowledgement* message;
- *download acknowledgement* – this message is a positive reply to the *download request* message;
- *block location list* – is sent from the tracker to a node, requesting the current list of blocks' location, this a reply to the *block location request* message.

Each node keeps two queues to process received messages. The former one is devoted to *download request* message. Each incoming request is placed in the queue and processed according to the selected decision strategy, which are described below. The latter queue contains *download acknowledgement* messages and analogously is managed according to selected strategy.

The P2P computing system we present in this paper is not centrally-managed, even though the system contains the tracker. Each node uses the same decision policy (strategy).

Decisions are taken by each participant individually according to its current knowledge including the following information:

- list of missing result blocks;
- transfer costs to other peers;
- index of the rarest block in the system.

The first information is updated by the peer according to completed operations (both processing and downloading of blocks). The other data is monitored and provided by the tracker.

We specify three main decisions for which we define policies in our system:

- *missing block selection* – the node that misses some blocks selects one of them for download. Here we propose two policies: *First-Missing* and *Rarest-Missing*. The former one assumes that the first missing block will be attempted to download. This works similarly to many P2P systems, where blocks are requested in order they combine into the desired file. Using the *Rarest-Missing* policy, the node checks which of its missing result blocks is the rarest result block in the system. Such block is requested to download. This information on the rarest block is provided by the tracker. However, the accuracy of the selection is limited to knowledge available at a node while selection is made. The idea of the rarest block is widely discussed in the context of P2P systems, e.g., in [19].
- *source node selection* – when a node needs to download the selected missing block, then it has to determine where to send the download request. This choice is made based on one of two following policies: *First-on-the-List* and *Cheapest-Owner*. The first method assumes that the first node from the list of desired block owners is selected. List of nodes having the desired block is obtained from the tracker and it is not ordered. The second policy (*Cheapest-Owner*) allows the node to analyze all nodes from owners list and select the node according to the settled criteria. This criterion may be cost, link speed, number of hops, etc. This way we get a system, which works with regard to optimizing one of many criterion factors.
- *request selection* – each node receives download requests from other nodes. As in many Peer-to-Peer systems, such requests are queued for later processing. This decision determines how such queue will be processed. Here we propose two policies: *First-Available* and *Cheapest-Available*. The former policy is the implementation of a FIFO queue, where the first received request is processed first. The *Cheapest-Available* policy assumes that node having bandwidth and other resources available

may select the request freely from all available in the queue. The criterion is the cost. If we assume, that our objective is be amount of data sent by requesting node, then this policy will resemble the tit-for-tat algorithm used in BitTorrent.

Decision strategies are described formally using in a pseudocode on Fig. 2.

Decision A: missing block selection

First-Missing:

Let Missing(v) return a set of blocks that are missing on node v. Let MinId(A) return an index of a block with a smallest value of the identification number included in set A.

```
int First-Missing(int v)
{
 return MinId(Missing(v));
}
```

Rarest-Missing:

We assume that function RarestBlock(A) finds the rarest block among blocks included in set A. This information is provided by the tracker, which has the global knowledge on the blocks' availability on each node.

```
int Rarest-Missing(int v)
{
 return RarestBlock(Missing(v));
}
```

Decision B: source node selection

First-on-the-list:

Function GetOwners(b) returns the set of nodes which own block b (thus may send this block to other node). This information is provided by tracker according to its knowledge. This set is not sorted and nodes are placed in order update messages arrive to tracker. Let FirstFromSet(A) return the first element from set A, and b denotes block which is to be download by node v.

```
int First-on-the-list(int b)
{
 return FirstFromSet(GetOwners(b));
}
```

Cheapest-owner:

Let function MinCost(v, A) return the id of node, from which transfer cost is smallest to node v. Returned node id is selected among node set A.

```
int Cheapest-Owner(int v, int b)
{
 return MinCost(v, GetOwners(b))
}
```

Decision C: request selection

First-available:

Let FirstRequest(v) return an index of a request which was put into the queue at earliest time among all request in the queue. Queue is owned by node v.

```
int First-Available(int v)
{
 return FirstRequest(v)
}
```

Cheapest-available:

Let MinRequest(v) return the id of a request, which was sent by a node having smallest transfer cost to node v. Request is chosen among requests present in the queue owned by node v.

```
int Cheapest-Available(int v)
{
 return MinRequest(v);
}
```

Fig. 2. Decision strategies in a pseudocode

The objective of the system is to minimize the operation cost of the system. We assume that the processing cost of each block includes the cost of the source block download. Therefore, we do not consider the cost of block transfer between the tracker and the node.

3. Simulation system

In this section we describe our concept of a discrete simulation system named CDSim (Computation-Distribution SIMulator). Unlike other network simulators (e.g., NS-2), our

system is designed to simulate both computations and distribution of obtained results. The architecture of P2P computing system presented in the previous section was implemented into a real simulation system. The simulator includes all elements of the computing system: nodes, tracker and decision policies. It also allows to use several types of network flows: unicast (results may be downloaded only from the node which computed it), anycast (traffic is routed through replica nodes) and pure Peer-to-Peer flow (results may be downloaded from any node which has the desired result block available).

We use the concept of time slots to model the time scale – thus our system is a discrete event system. The duration of each time slot is constant. One slot denotes the smallest considerable time period, during which a node may perform action, which does not require a reply from other elements. As described in the previous section, node resources are divided into *channels* – this way we can easily model the usability of a given resource. Technically, CDSim was written in C++ with use of STL and compiled using gcc (for Linux environment) and Visual Studio 2003 (for MS Windows environment). Parameters of simulations are set by a command line, detailed results of simulation are saved to a text file. Many levels of details logging are possible to be set. Input data (network structure, computational task, etc) is given as a text file with a standardized structure, what allows easy repetition of the simulation for one network. This file we call *network file* and it contains the following elements:

- number of nodes;
- number of replicas (used for anycast flow);
- cost of block computation for each block;
- cost of block transfer between each pair (logically as triangular matrix);
- number of iterations;
- task (blocks);
- upload and download speeds for each node;
- computing power of each node;
- number of channels.

The example networks are generated randomly according to fixed ranges of parameter values selected to model real network systems (e.g., access link capacity, processing power, transfer cost).

To illustrate the CDSim system we present a simple example using a Gantt graph (Fig. 3). The considered network consists of three nodes. Peer-to-Peer flows are used to deliver the results of computations. Each node is connected to the tracker with a pair of channels: upload (denoted as g_T) and download (denoted as h_T). Using the upload channel the node can send to the tracker one of three messages: *block location request* (denoted on the graph as L) *source block request* (denoted as C) and *tracker update* (denoted as U). In the opposite direction (on the download channel from the tracker to the node) the *block location list* message (denoted as L) can be transmitted as well as the source block (denoted as C). Moreover, each node has 3 processing channels (denoted as j_i, i = 1, 2, 3), 3 upload channels (denoted as g_i, i = 1, 2, 3) and 3 download channels (denoted as h_i, i = 1, 2, 3). Each channel is represented by one row in the graph. The considered computational project is divided into 7 tasks (blocks). A colored rectangle denote the operation on a particular channel (see the legend below the graph). The number placed in the center of each rectangle identifies the block, while the number is brackets shows the index of the corresponding node. White color denotes that the channel is idle. The processing channel can be occupied with processing of blocks. Download and upload channels can be used to exchange both signaling messages (*download request* and *download acknowledgement*) and data (result blocks) with other nodes.

Author's copy.

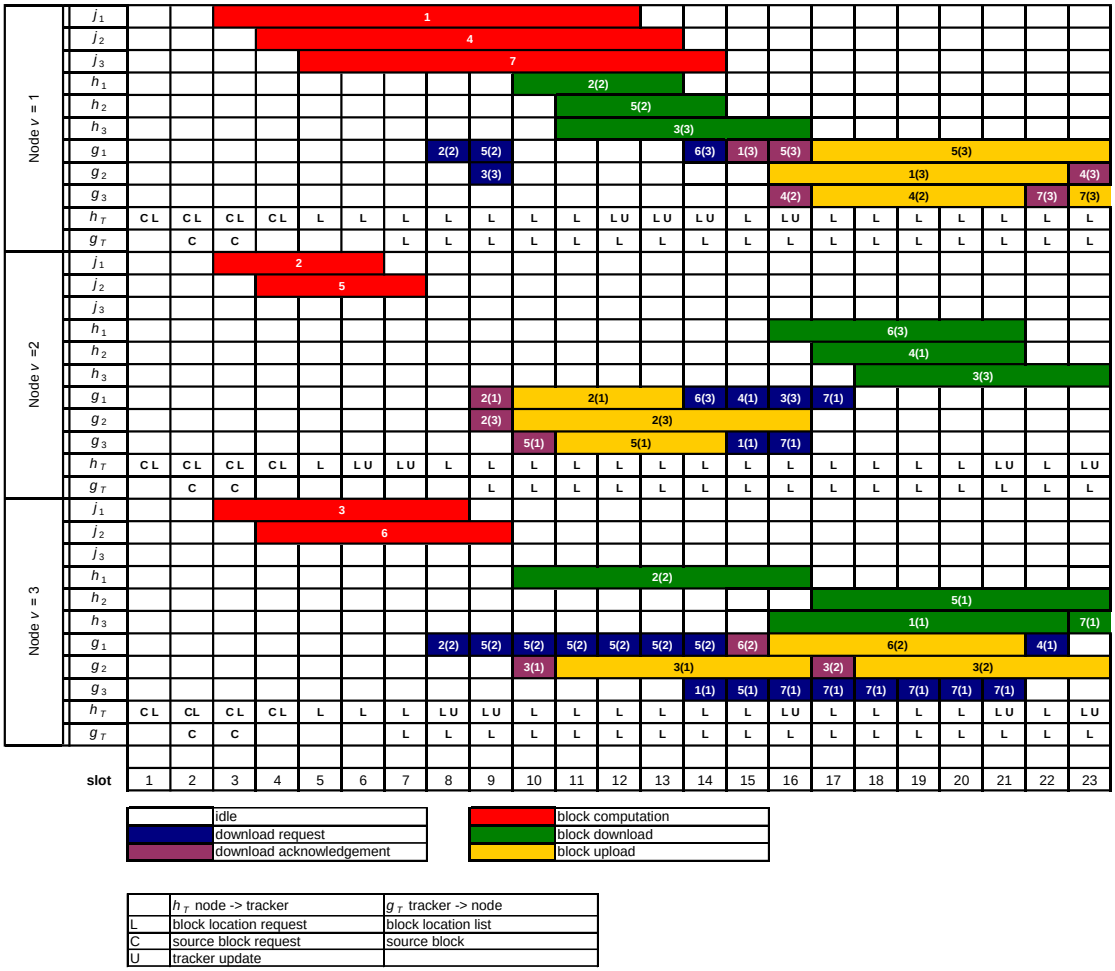

Fig. 3. A Gantt graph of ax example simulation

Each node sends to the trakcer the *source block request* and the *block location request* in slot $t = 1$ (denoted as C and L). The tracker replies with source blocks (denoted as C). The computation of the block can take different number of time slots according to computation power of each node. Until slot $t = 7$, nodes receive empty location list (*block location list*) from the tracker, since no blocks are yet computed. Starting from slot $t = 5$ nodes do not send *source block requests*, since all 7 source blocks are already assigned for computation. Block $b = 2$ is available to nodes in slot $t = 7$, therefore in slot $t = 8$ nodes $v = 1$ and $v = 3$ send *download requests* to node $v = 2$. Next, node $v = 2$ sends *download acknowledgements* back, so requesting nodes start downloading in slot $t = 10$. Block $b = 2$ is sent to nodes $v = 1$ and $v = 3$ with different speeds, according to the access link speeds of each pair of connected nodes (the transmission speed of each block is selected as the maximum possible transmission speed for the two communicating nodes). It is worth to notice, that even though node $v = 2$ uses decision 3 (*request selection*) to select requests from the queue, both requests are

answered. In case when node $v = 2$ would have only one upload channel available in time slot $t = 9$, only one request (selected according to the particular strategy) would be answered. This kind of situation may be observed in slot $t = 14$, where nodes $v = 1$ and $v = 2$ send download requests regarding block $b = 6$ to node $v = 3$. Node $v = 3$ selects only one request and sends the *download acknowledgement* to node $v = 2$. The example of the *source node selection* decision may be observed in time slot $t = 15$: node $v = 3$ wants to download block $b = 5$. According to its knowledge, block $b = 5$ is available for download from nodes $v = 1$ and $v = 2$. Using the given strategy, node $v = 3$ selects node $v = 1$ and sends there the *download request* message. When all download channels are busy, a node does not send further download requests – this situation may be observed in the case of node $v = 1$ during slots $t = 11...13$. Also, a node may be unable to send *download requests*, when it is busy with sending blocks to other nodes – see node $v = 2$ during slots $t = 11...13$.

4. Results

The simulator described in the previous section was applied to examine the proposed architecture of the P2P computing system. The performance metric reported in the experiments is the OPEX cost of the system including computing and transfer costs. We created 10 systems defined by the number of nodes, blocks, iterations and other parameters (Fig. 4). Other parameters (access link capacity, processing power, transfer cost and processing cost) were generated at random according to parameters of real overlay systems.

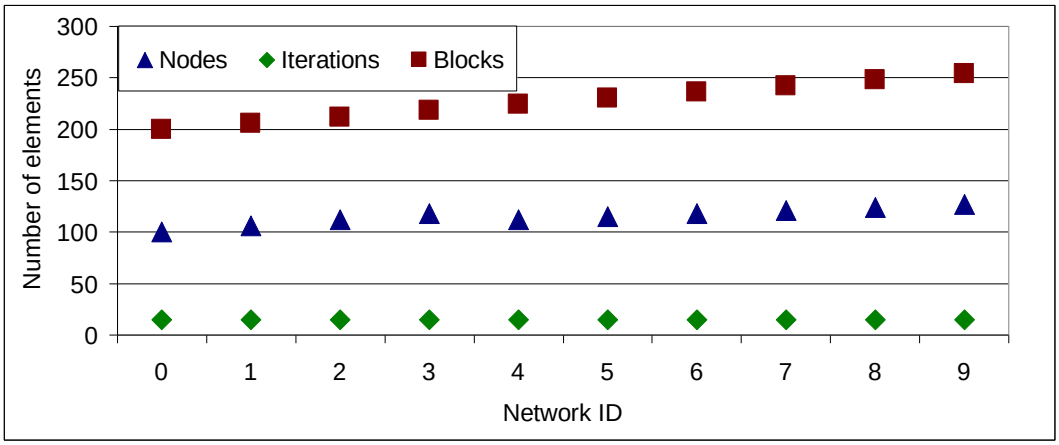

Fig. 4. Parameters of tested networks

The first experiment was focused on the evaluation of decision policies proposed in Section 2. Results show that the use of various policies has insignificant influence on the cost for unicast flow, because unicast is not much flexible in a terms of the data transfer.

Moreover, the *Rarest-Missing* policy in the case of the unicast flow and concurrent use of *Cheapest-Available*, *Cheapest-Owner*, *Rarest-Missing* policies set in the case of the anycast flow causes a starvation effect (some nodes were not able to get all result blocks in a given time). The P2P flow was fully unaffected by the starvation effect. Moreover, in the case of the P2P flow, the use of {*First-Missing*, *Cheapest-Owner*, *Cheapest-Available*} policies instead of {*Rarest-Missing*, *First-on-the-List*, *First-Available*} significantly decreases the cost, so the quality of delivered solution is much better (differences up to 60% for the P2P flow and up to 20% in the case of the anycast flow).

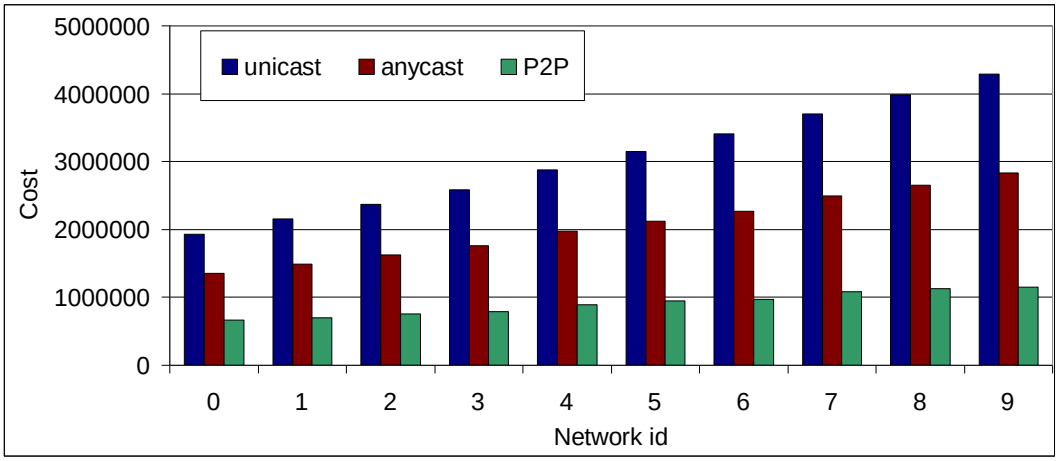

Fig. 5. The OPEX cost as a function of various types of flows

For further experiments we selected the following decision policies: *First-Missing, Cheapest-Owner, Cheapest-Available*. The next goal of simulations was to compare the OPEX cost for the following three types of network flows applied for data distribution: unicast, anycast and P2P. Fig. 5 reports the corresponding results for the same 10 systems. We can easily notice that the computing system using the P2P approach significantly outperforms the systems with unicast and anycast flows – the average reduction of the cost is 70% and 55%, respectively. Moreover, the unicast flow is the most sensitive to the growing number of nodes (57% difference between the minimum and the maximum cost), blocks and other parameters characterizing the size of the problem. This follows from the fact that the unicast flow is less flexible than P2P and anycast flows. The Peer-to-Peer flow with no additional restrictions for choice of sending nodes is the most resilient against problem growth and was able to keep a small increase of the cost for all tested networks (46% of difference between the minimum and the maximum cost). The P2P approach provides the best solution for all experiments, while the unicast flow always yielded the worst solution in terms of the cost.

Furthermore, we conducted experiments to evaluate our distributed system against standard, centralized computing systems. In the simulator system we implemented a computing system based on the BOINC architecture that uses one central server to collect the results data and next send the data to all participating nodes. We considered two cases related to the server location:

(i) the server location is not optimized, i.e. the distance to each node is the average of all other distances (denoted as C1);

(ii) the server is placed in the best location of the network minimizing the distance to other nodes (denoted as C2).

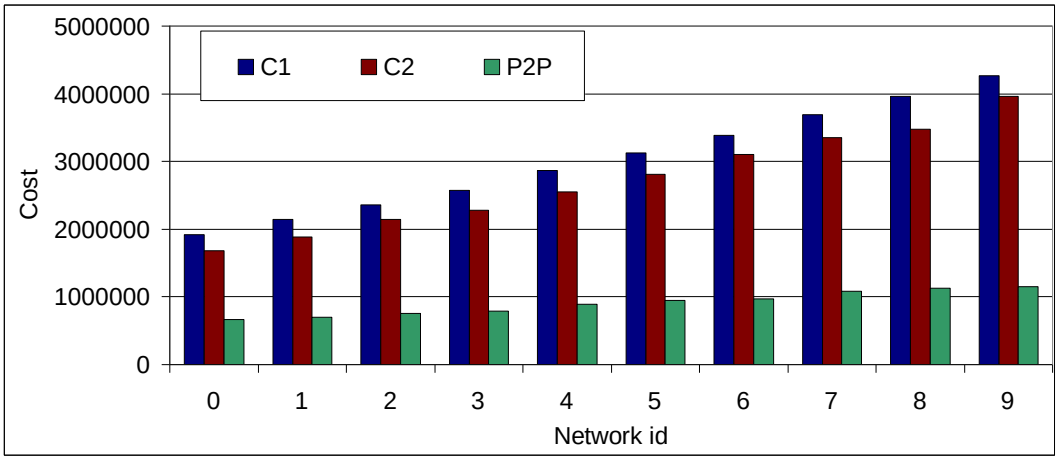

Fig. 6. The OPEX cost for distributed system with P2P flows (P2P), centralized system without optimization of the server location (C1) and centralized system with optimization of the server location (C2)

The results presented in Fig. 6 show that the distributed system considerably reduces the cost compared to centralized scenarios C1 and C2 – the average decrease is 70% and 66%, respectively. This experiment proves that our distributed architecture compared with the centralized approach can provide substantial gains in the terms of the OPEX cost.

The next research goal was to investigate the influence of joining new nodes into the system, keeping the amount of processed data at constant level. In this case, the cost increases for all network flows (the same policies set was used as in previous experiment). This relation is satisfied both for computation costs and transfer costs (consequently also for the total cost). Joining new nodes was repeated – and for each "new" network (previous network with a new node added) the simulation was executed. As the amount of data was constant, for each experiment we reached the point, where no feasible solution was yielded – because network bandwidth was too small to disseminate all results to all nodes and not so much data was available to satisfy fairness condition. The number of nodes at which the system stopped

returning proper solution was different for each network flow – the Peer-to-Peer flow always stopped at largest node number, unicast always quitted first, anycast was always in the middle. Similar results were observed for another experiment, in which we increased the amount of data in the network, keeping all other parameters constant (including processing power, network bandwidth, etc). The P2P approach always reached highest number of blocks processed. Thus, the P2P flow yielded the smallest cost and was resistant for undesirable phenomena such as the node starvation. Moreover, the P2P approach provided the best performance in the case of joining new nodes to the network and increasing the amount of data present in the system.

5. Related work

Grid systems are most often considered through the aspect of scheduling, as a significant issue for grid efficiency. The authors of [20] proposed the policy of assigning resources to grid participants. The problem was formulated as a variant of a multichoice multidimensional knapsack problem. Described policies were investigated and proven to be efficient. Static modeling of grid systems was proposed in [5], which also describes grid resource management in scope of authorization and monitoring. Other approach to grid management was presented in [4] and considered resource monitoring, resource scheduling, and usage of network links and storage resources. The authors of [10] introduced a distributed computing system, which is dedicated to render images. In contrary to previous approaches – which required presence of much amount of data on each participating node – the proposed system uses a division of images into primitives, which are then replicated on many nodes. Results of rendering are sent to the central node, where they are combined into final result image. Hughes and Walkerdine [11] presented a distributed computing system designed to process video files using Peer-to-Peer structure. This system also uses central node to produce the final result video file.

Most of distributed computing systems assume that source blocks (created as result of division of input task into blocks) are fully independent – what means that particular node does not need to know about other blocks to compute block assigned to itself. GTapestry model presented in [21] allows for using relations between blocks – they may be dependent between each other. Nodes are classified into groups, which then compute groups of blocks connected with relations. Nodes may communicate inside groups (intra-communication) or between groups (inter-communication), what allows managing blocks' dependencies to get the final result.

Concepts of distributed computing systems and Peer-to-Peer networks share many common ideas. Both approaches consist of many elements performing the specified roles. The concept of merging computation and Peer-to-Peer ideas was presented in [22] – authors described differences, approaches and problems occurring in these two kinds of systems. Fox *et al.* described the conjunction of grid and Peer-to-Peer systems, also introducing additional communications layer using XML to describe data and messaging between elements of the system.

Network simulators most commonly use a similar form of internal architecture, based on discrete simulation idea. This approach assumes the use of an internal clock, which divides simulation time into a set of time slots. The length of the time slot determines the accuracy of the modeling. Discrete event simulator contains the following elements: internal clock, events, random value generator and monitoring modules [23]. Internal clock handles all issues related to step-by-step modeling – either one or more events are possible to happen during one time slot. Events model all real occurrences that are considered during simulation – they may be distributed using simulator's event generator and stored in event queue [23] or issued by any other network element, such as servers, peers, etc. Some of simulation details are often based on random generated values – as many things in real systems occur influenced by random factors and real systems are never fully deterministic. Examples of such details are:

- packet latency,
- best peer selection,
- order of messages sent at the same time and arriving to same queue etc.

Thus, each discrete simulator contains a pseudo-random number generator, and its randomness quality is very important in terms of the quality of the whole simulation. It is vital to underline how the simulation ends, to avoid the system to run forever – and such condition is most often required by simulation systems. The other approach to simulation is the *event-driven simulation*, which is based on the idea, that the time of simulation advances only when simulation events occur [24] (in the case of the discrete simulation, time slots advance no matter events occur or not). This approach have not become very popular, as discrete event simulation emerged to provide highly satisfying results. However, there are extensions of discrete simulation proposed, such as RTNS [25] or a hybrid approach for timing [26], which are introducing event-driven ideas to discrete simulation and are used to model wireless sensor networks.

Many authors use popular simulation tools such as NS-2 [27] and OPNET [28]. These systems focus on research in following fields: network topologies, network protocols, wired

and wireless networks (including ad-hoc and sensor networks). NS-2 simulator is often categorized as a ‘packet network simulator’, as it was designed to simulate this kind of computer networks. NS-2 requires the user to implement his logic, which operates on NS-2 framework in the area of protocols, network types, network elements and traffic models [27]. Dedicated Otcl language is proposed to model the desired architecture. Many papers propose and discuss the extensions for NS-2 simulator, such as single and multiprocessor embedded systems, sensor function models (SensorSim [29], RTNS [25], TOSSIM [26]), multicast flow [30], and many others. OPNET is a commercial product intended to model network flows, applications, devices, protocols and many other network elements. It provides many protocols in a shape of source code, object-oriented and hierarchical modeling and graphical modeling tools. NS-2 and OPNET simulators are similar by idea and were compared by authors of [31]. Authors of [32] compare results of a realtime scheduler and a simulator of a non-realtime scheduler. Results showed that pure discrete simulation approach may be successfully extended to make simulated network less ideal and therefore also closer to the real networks.

6. Concluding remarks

The distributed computation is a very efficient and promising approach to process large amounts of data without generating high costs. Private volunteers around the world are willing to contribute with their resources that combined into one virtual structure constitute a large processing power available for computing common task. Traditional grids assume central management, what is not efficient in the case when all (or most) system’s participants are willing to get the complete result. In this paper we have proposed a distributed computing system that optimizes both computations and transfers.

The system can use three kinds of network flows for data distribution: unicast, Peer-to-Peer and anycast. In this paper we have described the structure of the proposed system, suggested policies and technical details of the developed realtime simulation system. The use of discrete realtime simulation is a very valuable approach to network optimization. It often overcomes disadvantages of the static optimization. Unlike other network simulators, our simulating system CDSim introduces possibility not only to model the network traffic, but also the process of distributed computing. In the current form, CDSim allows to investigate three types of network flows (unicast, Peer-to-Peer and anycast). Many network parameters are considered and possible to investigate, also helping tools (like network generator) provide efficient way to perform network simulations. We have focused not only on simulator itself, but also on surroundings – input and output data, and possibility to make whole system

efficient and flexible. We have used the CDSim to make experimental research, which showed that the use of the P2P flow allows getting much lower OPEX costs comparing to unicast and anycast flows, proving that flow flexibility is very important in distributed computation systems. Moreover, we have shown that our distributed system significantly outperforms the centralized approach.

As the future work, we propose to extend the computation system with new constraints – like varied number of nodes and replica statuses changed during the simulation. Other future directions are new decision policies that can be evaluated by using the CDSim simulator.

Acknowledgments

This work is supported by The Polish Ministry of Science and Higher Education under the grant which is being realized in years 2010-2013.

References

[1] F. Travostino, J. Mambretti, G. Karmous Edwards, *Grid Networks Enabling grids with advanced communication technology*, Wiley, 2006.

[2] R. Buyya, *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, Ph D Thesis, School of Computer Science and Software Engineering, Monasch University, Melbourne, 2002.

[3] Ruay-Shiung Changa, Ming-Huang Guo, Hau-Chin Lin, *A multiple parallel download scheme with server throughput and client bandwidth considerations for data grids*, Elsevier Future Generation Computer Systems 24, Vol. 24, No. 8, 2008, pp. 798–805

[4] K. Krauter, R. Buyya, and M. Maheswaran, *Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing*, International Journal of Software: Practice and Experience (SPE), Vol. 32, No. 2, 2002, pp. 135–164

[5] J. Nabrzyski, J. Schopf, J. Węglarz (eds), *Grid resource management: state of the art and future trends*, Kluwer Academic Publishers, Boston, 2004.

[6] N. Fujimoto, K. Hagihara, *A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks*, Proceedings of the 2004 International Symposium on Applications and the Internet Workshops (SAINTW'04), 2004, pp. 674–680.

[7] D. Anderson, *BOINC*: *A System for Public-Resource Computing and Storage*, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4–10.

[8] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, *SETI@home: An Experiment in Public-Resource Computing*, Communications of the ACM, Vol. 45, No. 11, 2002, pp. 55–61.

[9] Statistics of projects based on BOINC framework: http://boincstats.com .

[10] R. Samanta, T. Funkhouser, K. Li, *Parallel Rendering with K-Way Replication*, In Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics, 2001, pp. 75–84

[11] D. Hughes, J. Walkerdine, *Distributed Video Encoding over a Peer-to-Peer network*, In the proceedings of PREP 2005, Vol. 1, 2005.

[12] S. Draves, *The Interpretation of Dreams, An Explanation of the Electric Sheep Distributed Screen-Saver*, http://electricsheep.org/.

[13] A. Legout, G. Urvoy-Keller, P. Michiardi, *Understanding BitTorrent: An Experimental Perspective. Technical Report*, INRIA-00000156, 2005.

[14] R. Ahuja, J. Magnanti, J. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Englewoof Cliffs NJ, 1993.

[15] M. Pióro, D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*, Morgan Kaufman Publishers 2004.

[16] R. Steinmetz, K. Wehrle (eds.), *Peer-to-Peer Systems and Applications*, Lecture Notes in Computer Science, Vol. 3485, 2005.

[17] J. Han, D. Watson, F. Jahanian, *Enhancing end-to-end availability and performance via topology-aware overlay networks*, Computer Networks Vol. 52, No. 16, 2008, pp. 3029–3046

[18] D. C. Vanderster, N. J. Dimopoulos, R. Parra-Hernandez, R. J. Sobie, *Resource allocation on computational grids using a utility model and the knapsack problem*, Future Generation Computer Systems, 25, 2009, pp. 35–50.

[19] S. Zhang, S. Zhang, X. Chen, X. Huo, *Cloud Computing Research and Development Trend*, Second International Conference on Future Networks, 2010, pp. 93–97.

[20] F. Mathieu, J. Reynier, *Missing Piece Issue and Upload Strategies in Flashcrowds and P2P-assisted Filesharing*, Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services, 2006, pp. 112–118.

[21] Hai Jin, Fei Luo, Qin Zhang, Xiaofei Liao, Hao Zhang, *GTapestry: A Locality-Aware Overlay Network for High Performance Computing*, Proceedings of 11th IEEE Symposium on Computers and Communications, 2006, p. 76–81.

[22] R. Subramanian, B. Goodman, *Peer to Peer Computing: The Evolution Of A Disruptive Technology*, Idea Group Publishing, 2005.

[23] M. Bumble, L. Coraor, *Architecture for a non-deterministic simulation machine*, Proceedings of the 1998 Winter Simulation Conference, 1998, pp. 1599–1606.

[24] I. Jawhar, *A Flexible Object-Oriented Design of an Event-Driven Wireless Network Simulator*, Wireless Telecommunications Symposium, 2009, pp. 80–86.

[25] P. Pagano, M. Chitnis, G. Lipari, *RTNS: an NS-2 extension to Simulate Wireless Real-Time Distributed Systems for Structured Topologies*, Proceedings of the 3rd international conference on Wireless internet, 2007.

[26] A. Lalomia, G. Lo Re, M. Ortolani, *A Hybrid Framework for Soft Real-Time WSN Simulation*, 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, 2009, pp. 201–207.

[27] E. Altman, T. Jiminez, *NS Simulator for beginners*, NS-2 documentation, 2003.

[28] Webpage of OPNET project: http://www.opnet.com .

[29] S. Park, A. Savvides, M. B. Srivastava, *SensorSim: A Simulation Framework for Sensor Networks*, International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems, 2000, pp. 104–111.

[30] S. Penz, *Wireless Multicast Support for the NS-2 Emulation Environment*, Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2007, pp. 267–273.

[31] M.B. Jemaa, N. Baccour, H. Kaaniche, *A comparative Study of two Ad Hoc Network Simulators*, Septie`mes journées scientifiques des jeunes chercheurs en génie électrique et informatique (GEI 2007), Monastir, Tunisia, 2007.

[32] Ju-Young Shin, Jong-Wook Jang, Jin-Man Kim, *Result based on NS2, Simulation and Emulation Verification*, International Conference on New Trends in Information and Service Science, 2009, pp. 807–811.