Keywords: distributed computing, unicast, heuristic algorithm

Grzegorz CHMAJ*

HEURISTIC ALGORITHM FOR CLIENT-SERVER DISTRIBUTED COMPUTING

The distributed processing systems contain many nodes connected into one logical structure that is able to perform the processing of given task. The task is submitted by the system operator – and from his point of view the system is inputted with a task, computing layer processes the task and the result is returned by the system. Such layer-oriented approach requires to design the operational mechanisms of the lower layers – computation and communication layers. In this paper the unicast approach is studied – the communication between nodes goes through a central node – server: the operational algorithms are defined and the experimentation results are presented.

1. INTRODUCTION

The distributed processing systems are utilizing the computational power of multiple devices – to gain the significant processing power used then to compute the given task. Such systems can have various scale – from system-on-chip, through multiprocessor local system, to large scaled ones, incorporating thousands of machines spread throughout the whole world. The public distributed processing systems are the ones that are most known in the world. They are usually based on BOINC[1] software and are called public because every internet user can install the local client on his home computer and then join the selected project and contribute his local processing power. The most famous BOINC-based project is Seti@home – where radio signals captured by radio telescope are sliced into fragments and processed by the system to look for the extra-terrestrial intelligence. In this paper, the unicast approach is studied – the distributed system contains many nodes, but directs

* Department of Electrical and Computer Engineering, University of Nevada Las Vegas, USA
its communication through the central node (server). The algorithm for such operation is presented, along with the experimentation results.

2. SYSTEM DESCRIPTION

Public distributed computing system – is a structure built with many machines (called nodes) connected into one structure, which may be considered as one virtual machine with big computing power. Each node is the machine connected into the computing system through a link. Connection link has upload and download limits, node has also processing power limited. Node introduces its computing power and serves the same role as each other system’s machine. The internal structure of the system might contain some nodes of special functions though. System receives computational task, which is then computed using the computing system. This task is divided into uniform fragments called source blocks. These blocks are computed at system nodes, the result of computation we call result blocks. One source block computed – results in one result block. For the sake of simplicity – the size of each source block is the same, the same rule is true for result blocks. To compute each source block, the same amount of computing power is required. Moreover, source blocks are computationally independent – i.e. one block may be computed without any knowledge about rest of source blocks. These assumptions model rules used in popular public computing such as Seti@home or Climate Prediction. In Seti@home project: input task is the long lasting radio signal in a form of sound “file”. This signal is divided into fragments having the same length (source blocks) – thus having same computational power requirements. Each fragment is analyzed independently, result is sent back to central node. The same size of result blocks we should relate to transport system, such as popular Peer-to-Peer protocol BitTorrent. It divides the file to fragments having same size (typically 256kB) – what is the analogy to our result blocks size equality.

There is logical direct connection between each two nodes. This connection is created in lower layer of the overlay network, so it is not visible to layer at which we place our model. This concept is often used in literature [2], [3] as well in daily networking – the most popular overall network the Internet. The time scale of system processing is divided into time slots, which we call iterations. Each time slot may be considered as time period of given duration expressed in seconds. Each iteration has the same length. During each iteration, nodes may transfer result blocks between them, but the information about blocks available at nodes is updated with iteration change (i.e. blocks downloaded by node v in iteration t can be sent by v to other nodes not earlier than during iteration t + 1). The length of iteration may be considered as
follows: if total size of data sent by node \( v \) is 256kB, and upload link speed is 128kb/s, then duration of iteration is 16s (256kB/128kb/s). The idea of using time slots in static modeling was also used in [4], [5], and many others.

The system considered in this paper performs the distributed processing for the given task, however the communication between nodes is done through a special node – called server. This way this system is called “client-server distributed processing system”.

3. RELATED LITERATURE

Unicast type of communication, usually referred as client-server architecture is a widely used mechanism of communication. However it introduces limits in cases when massive amounts of the same data needs to be delivered to multiple nodes. [6] studies the routing algorithm with the regard of bandwidth limitation. Authors model the unicast approach and consider both single unicast and group unicast communication. Due to unicast limitations, another communication models are studied and compared. Authors of [7] describe the allocation of the bandwidth in unicast and multicast flows. They also provide the evaluation of bandwidth available and describe the proposed policies for multicast approach.

The base of many distributed processing system – BOINC – was described in [1]. Author described some details of the system, mentioned the popular projects based on this platform, and compared it to the grid computing. Multiple aspects of BOINC-based public distributed computing were evaluated in [8]. Authors included system architecture, infrastructure, applicability, security, result validation, client architecture and some others. The aspect of motivating the volunteers was described as well. [9] presents the specific application of public computing – finding protein binding sites. The work related to a project development, early stages challenges and implementation details were provided. The real-time variation of the BOINC platform was shown in [10] – such elements of real-time system as deadline timer and parameter-based admission control were added. Authors also describe extensions and their relation to original BOINC.

4. HEURISTIC ALGORITHMS

In order to solve the problem stated, the following heuristic algorithm was developed. It consists of two main parts: UH1 – the allocation of source blocks, UH2 – the results distribution. The algorithms are offline – the input data is known at the time
the algorithm is executed. The algorithm notation uses the following elements: \( b = 1, 2, \ldots, B \) – blocks, \( t = 1, 2, \ldots, T \) – time slots, \( v, w = 1, 2, \ldots, V \) – network nodes, \( c_v \) – cost of block computation at node \( v \), \( k_{vw} \) – cost of block transfer between nodes \( w \) and \( v \), \( p_v \) – computation limit of node \( v \), \( d_v \) – download limit of node \( v \), \( u_v \) – upload limit of node \( v \). Two binary variables are used:

\[ x_{bv} = \begin{cases} 1 & \text{when block } b \text{ is computed at node } v; \\ 0 & \text{otherwise} \end{cases} \]

\[ y_{bwvt} = \begin{cases} 1 & \text{when block } b \text{ is transferred from node } v \text{ to node } w \text{ in iteration } t; \\ 0 & \text{otherwise} \end{cases} \]

The algorithms are defined the following way:

**UH1 – source blocks’ allocation**

0. Assign \( a_v \) blocks to each node \( v = 1, 2, \ldots, V \):

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

1. If there are still not allocated blocks (\( \sum_v a_v < B \)), go to step 2, otherwise exit algorithm

2. For each node \( v = 1, 2, \ldots, V \) compute score \( e_v \) using formula (2.14):

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

3. Determine maximum score among all score values: \( e_{\max} = \max_{v=1,2,\ldots,V} (e_v) \)

4. Compute score gap \( g_v \) for each node using (2.16):

\[
g_v = \begin{cases} 0 & \text{when } p_v - a_v = 0 \lor a_v(V - 1) \geq u_v T \\ \frac{e_{\max} - e_v}{e_v} & \text{otherwise} \end{cases}
\]

Put all \( g_v \) values in array and sort it descending.

5. Allocate blocks to nodes:
   a. Point to first element of array (this element identifies node \( v \) having highest gap \( g_v \))
   b. Assign \( a'_v \) blocks to node \( v \) using formula (2.17):

\[
a'_v = \begin{cases} \frac{u_v T}{V - 1} - a_v & \text{when } p_v > \frac{u_v T}{V - 1} \text{ and } B - \sum_w \sum_w x_{bw} + a_v - \frac{u_v T}{V - 1} \geq 0 \\ p_v - a_v & \text{when } p_v \leq \frac{u_v T}{V - 1} \text{ and } B - \sum_w \sum_w x_{bw} + a_v - p_v \geq 0 \\ B - \sum_w \sum_w x_{bw} & \text{otherwise} \end{cases}
\]
c. If there are still non-allocated blocks, then point to next element of \( g_v \) array (identifying node \( v \) having gap \( g_v \)) and go to step 5b. Otherwise finish algorithm.

**UH2 – result blocks’ distribution**

0. Create list \( L_v \) containing all nodes and list \( L_b \) containing all result blocks.

1. Let \( f_v \) denote the pointer to element on list \( L_v \) and \( f_b \) denote the pointer to element on list \( L_b \). Let \( l_{vn} \) denote \( n \)-th element on list \( L_v \), and \( l_{bn} \) denote \( n \)-th element on list \( L_b \). Set iteration \( t = 1 \).
   a. Set pointers to first elements of lists: \( f_v = l_{v1} \) and \( f_b = l_{b1} \)
   b. Check if node \( v \) at position \( f_v \) on list \( L_v \) is able to make download \((\sum_y \sum_w y_{bvvt} < d_v)\). If yes – then go to point 1c), otherwise go to point 1g)
   c. Check if block \( b \) at position \( f_b \) is present on node \( v \) identified by \( f_v \). If not, go to 1d) otherwise go to 1f)
   d. Check if node \( w \) which satisfies condition \( x_{bwv} = 1 \) is able to make upload \((\sum_v \sum_y y_{bwvt} < u_w)\). If yes – then go to 1e) otherwise go to 1f)
   e. Send block \( b \) from node \( w \) to node \( v \) \((y_{bwvt} = 1)\). Increase pointer \( f_b \) by one (so it identifies next element on list \( L_b \)). Go to 1g).
   f. Increase \( f_b \) by one. If block \( b \) at position \( f_b \) was already considered in point 1c) for node \( v \) identified by \( f_v \), then go to 1g), otherwise go to 1c)
   g. Increase \( f_v \) by one. If there was no block transfer since element \( f_v \) was previously analyzed, then go to 1i), otherwise go to 1b)
   h. If every node possesses all result blocks, then exit algorithm. Otherwise go to 1i)
   i. Set pointers to first elements of lists: \( f_v = l_{v1}, f_b = l_{b1} \). Increase \( t \) by one (what means switching to next iteration). If \( t > T \) (there are no iterations left), then exit algorithm, otherwise go to step 1b)

Complete algorithm for unicast flow may be described in three following steps:

**UHA – algorithm for unicast flow.**

0. Allocate blocks using algorithm UH1
1. Perform blocks’ computation
2. Distribute result blocks to nodes using algorithm UH2.
4. EXPERIMENTS

In order to evaluate the quality of the algorithms described above, the comparison between heuristics and optimal solutions were done. The optimal solutions were produced using MIP models implemented in CPLEX optimizer, the time of CPLEX experiment was limited to 3600 seconds. Due to complicity of researched problems, optimal solution was achieved only for small networks with relatively small number of blocks and iterations. Increasing problem size results with firm growth of computation time (for optimal solutions) and memory need, thus even increasing time limit for CPLEX does not affect the quality of solutions significantly. The quality of a solution is measured using the function:

\[ F = \sum_{b} \sum_{v} x_{bv} c_v + \sum_{b} \sum_{v} \sum_{w} y_{bwv} k_{wv} \]

Function \( F \) sums up all the network costs and processing costs – and models the electrical energy consumption. The researched 300 networks had the following parameters:

<table>
<thead>
<tr>
<th>parameter</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of nodes</td>
<td>3 – 17</td>
</tr>
<tr>
<td>number of iterations</td>
<td>2 – 9</td>
</tr>
<tr>
<td>number of blocks</td>
<td>3 – 26</td>
</tr>
</tbody>
</table>

CPLEX returned 243 optimal solutions, for 51 networks its solution was classified as feasible, and for 6 networks there was no solution at all. We will focus on networks, for which CPLEX provided optimal solution – which we denote as UOA (Unicast Optimal Algorithm). Investigation showed, that comparing 243 optimal UOA solutions with related UHA solutions, average \( D_{\text{UOA}} \) (difference between UHA and UOA) was 0% (242 cases where \( F_{\text{UOA}} = F_{\text{UHA}} \) and one case where CPLEX provided optimal solution, while UHA solution did not provide any result). Due to unicast specific nature – the communication can be done only through the server – the optimization concerns the source blocks allocation (with regard to the various parameters of nodes). The experiments results show, that UHA algorithm performs very well, also the time of the execution was very low (below 1 second, while CPLEX needed much more time for larger networks). However the algorithm is not ideal – in one case UHA was unable to provide any result, while we know that the solution exists, as it was produced by CPLEX.

The relation between the number of nodes \( V \) and the cost of system operation was also evaluated. As presented on Fig. 1. the cost of system operation \( F \) increases as the
number of nodes $V$ increases. The relation is almost linear. This is caused by a specific property of the system presented:

$$x_{bv} + \sum_b \sum_{y_{bv}} = 1 \quad b = 1, 2, \ldots, B \quad v = 1, 2, \ldots, V$$

It says, that each node needs to receive all the results produced by the system. With this requirement in mind, it is easy to explain that each node added to the system generates the additional traffic of delivering all the results to the node added.

Fig. 1. Relation between $V$ and $F$

6. CONCLUSIONS

In this paper, the distributed processing system with client-server network approach is described, along with the operational offline algorithm. The research results show, that the algorithm performs very well in terms of the metrics defined and time required to obtain the solution. The further work concerns another communication models – peer-to-peer and anycast: the design of the algorithms and their time/cost performance compared to optimal solutions. Finally, the evaluation of properties and the cost of operation should be done to compare communication models between themselves – when applied to the same processing problem.
REFERENCES


