A Network Simulation Tool for Task Scheduling

Ondřej Votava

Dept. of Computer Science, Czech Technical University, Karlovo nám. 13, 121 35 Praha 2, Czech Republic

Corresponding author: votavon1@fel.cvut.cz

Abstract

Distributed computing may be looked at from many points of view. Task scheduling is the viewpoint, where a distributed application can be described as a Directed Acyclic Graph and every node of the graph is executed independently. There are, however, data dependencies and the nodes have to be executed in a specified order. Hence the parallelism of the execution is limited. The scheduling problem is difficult and therefore heuristics are used. However, many inaccuracies are caused by the model used for the system, in which the heuristics are being tested. In this paper we present a tool for simulating the execution of the distributed application on a "real" computer network, and try to tell how the execution is influenced compared to the model.

Keywords: task scheduling, DAG scheduling, simulation, network simulation.

1 Introduction

Heterogeneous computation platforms have become very popular in the past decade. They are cheap and easy to construct and offer good computation power. Compared to parallel computers, distributed systems offer better price-to-power ratio. However, the properties of distributed systems are different. Communication is provided by a high-speed network which is still slow in comparison with the specialized networks used in parallel systems [15]. Mainly, communication leads to the need to modify traditional parallel algorithms into distributed algorithms [25, 14, 19].

Task scheduling is one of many approaches used for distributed algorithms. The idea is simple. Let us take an application: this application consists of several parts that may be executed independently. These part can then be computed on different computers concurrently and the application can be speeded up. Task scheduling tries to answer which parts should be computed on which computers and when, so that the computation time is minimized.

The structure of the paper is as follows. In the next section we describe the problem of task scheduling itself, and show several approaches that are wide used for solving the problem. At the end of the next section we show the network-related problem of the simplified models that are used. Section 3 then describes the simulation tool that we used for making measurements, and in section 4 we show some interesting results obtained from the simulations.

2 Task scheduling

The application that is to be scheduled can be described as a Directed Acyclic Graph (DAG), i.e. $AM = (\mathbf{V}, \mathbf{E}, \mathbf{B}, \mathbf{C})$, where:

$\mathbf{V} = \{v_1, v_2, \ldots, v_v\}$, $|\mathbf{V}| = v$ is the set of tasks, task $v_i \in \mathbf{V}$ represents the piece of code that has to be executed sequentially on the same machine;

$\mathbf{E} = \{e_1, e_2, \ldots, e_e\}$, $|\mathbf{E}| = e$ is the set of edges, edge $e_j = (v_k, v_l)$ represents data dependencies, i.e. task v_l cannot start computation until the data from task v_k have been received, task v_k is called the parent of v_l, v_l is called the child of v_k;

$\mathbf{B} = \{b_1, b_2, \ldots, b_v\}$, $|\mathbf{B}| = v$ is the set of computation costs (e.g. number of instructions), where $b_i \in \mathbf{B}$ is the computation cost for task v_i;

$\mathbf{C} = \{c_1, c_2, \ldots, c_e\}$, $|\mathbf{C}| = e$ is the set of data dependency costs, where $c_j = c_{k,l}$ is the data dependency cost (e.g. amount of data) corresponding to edge $e_j = (v_k, v_l)$.

A task which has no parents or children is called an entry task or an exit task, respectively. If there is more than one entry/exit task in the graph a new virtual entry/exit task can be added to the graph. Such a task would have zero weight and would be connected by zero weight edges to the real entry/exit tasks.

The application has to be scheduled on a heterogeneous computation system (CS) which can be described as a general graph, but there is one very important restriction. The graph represents a connection structure, and even though there may be no direct connection of two computation nodes there must be an edge between all of the nodes that are able to communicate. This restriction leads to the observation that the CS is always a complete graph. The computation system can then be described as $CS = (\mathbf{P}, \mathbf{Q}, \mathbb{R}, \mathbb{S})$, where:

$\mathbf{P} = \{p_1, p_2, \ldots, p_p\}$, $|\mathbf{P}| = p$ is a set of the computation nodes;

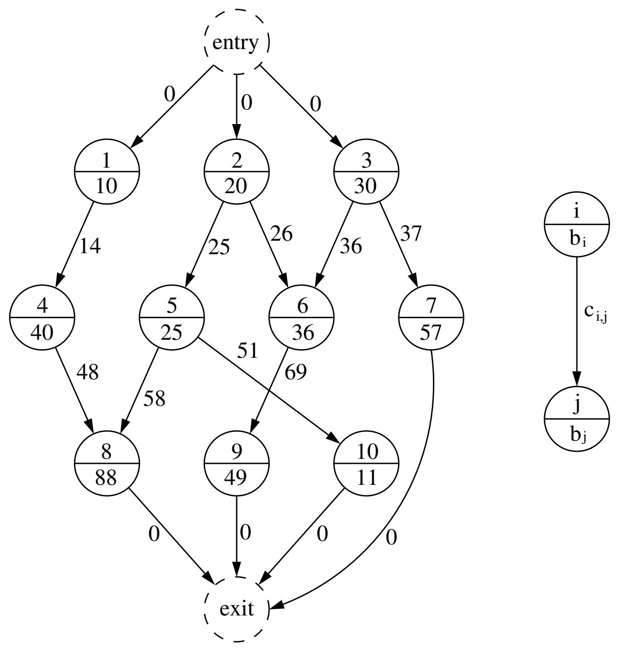


Figure 1: Application model described as a DAG.

Q $= \{q_1, q_2, \ldots, q_p\}, |\mathbf{Q}| = p$ is the set of speeds of computation nodes, where q_i is the speed of node p_i;

$\mathbb{R}$ is a matrix describing the communication costs, the size of $\mathbb{R}$ is $p \times p$;

$\mathbb{S}$ is a matrix used to describe the communication startup costs; it is usually one-dimensional, i.e. its size is $p \times 1$.

Scheduling is connected to the specific application and the specific *CS*. The computation time $t_{i,j}$ of task v_i on a node p_j can be calculated using equation

$$t_{i,j} = \frac{b_i}{q_i}. \tag{1}$$

When thinking of static scheduling, matrix $\mathbb{W}$ can be used. $\mathbb{W}$ contains information on the computation times for all of the tasks on every node, i.e. the size of $\mathbb{W}$ is $v \times p$.

The scheduling algorithm has to take into account the communication delay. The duration of transfer of the edge e_i from node p to node q is then defined as

$$t(i)_{p,q} = \mathbb{S}[p] + c_i \cdot \mathbb{R}[p][q]. \tag{2}$$

2.1 Scheduling algorithms

The problem of task scheduling is claimed to be NP-complete [21, 7] and therefore intensive research in heuristics has been done. The heuristics can be divided into several categories, the common criterion being knowledge of when the schedule is created. If the schedule is computed *before* computation of the application begins, i.e. if the schedule is known a priori, the heuristic is called *static* or *offline*. In contrast, when the schedule is computed as a part of the computation of the application, the heuristic is called *dynamic* or *online*.

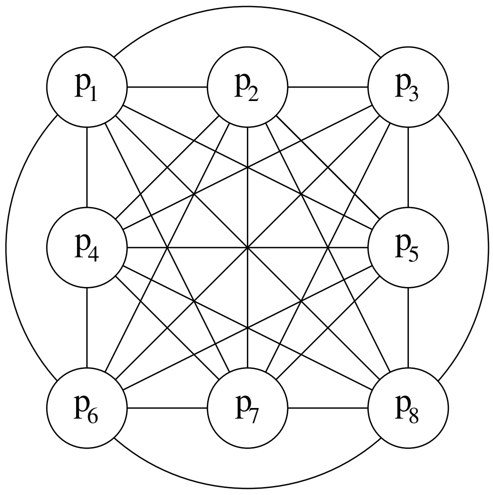
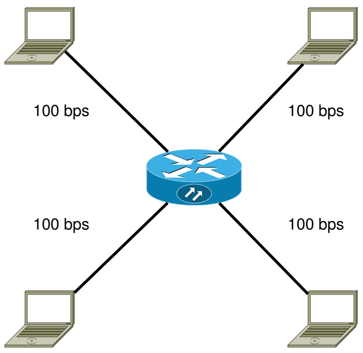


Figure 2: The computation system described as a complete graph.

Both static and dynamic algorithms have been proposed in the literature. For a heterogeneous computation platform, an example of a very well-known static algorithm is HEFT [20]. The main idea of HEFT is to order the tasks in a list and to assign the tasks which are ready to the computer which minimizes the execution time for the task. Another algorithm proposed in [20] is CPOP. This algorithm finds a critical path and minimizes the execution of tasks which are in the path. The quality of the schedule is then very dependent on how the critical path was created. CPOP has slightly worse computation complexity than HEFT, but scheduling quality results are close.

The idea of creating a list of tasks ordered in a specific manner is common for a whole group of scheduling heuristics. They are called *list scheduling* algorithms. Modifying HEFT in such a way that some tasks can be duplicated, we obtain the algorithm presented in [8]. This algorithm can then be applied to a specific context of cluster-based computation systems, and the results that it achieves are very good [11]. Many other algorithms have been published. Some were summarized in [2], and many other, which are focused on homogeneous computation system, are compared in [13].

Unlike static algorithms, dynamic algorithms are usually used to schedule more than one application at a time. However, there are some exceptions. For example, [24] schedules several applications in a static way and compares several attitudes that are permissible for this problem. A semi-dynamic attitude is



$$\mathbb{R} = \begin{pmatrix} 0 & 0.01 & 0.01 & 0.01 \\ 0.01 & 0 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0 & 0.01 \\ 0.01 & 0.01 & 0.01 & 0 \end{pmatrix}$$

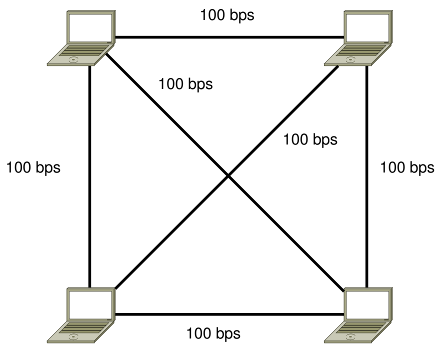


Figure 3: A real network (left), its parameters, and the false representation from matrix $\mathbb{R}$ (right).

described in [1], where the tasks are scheduled statically, but there is a global application structure which contains all of the applications, and this global structure changes when a new application arrives in the system. A completely different attitude to the dynamic algorithm is presented in [17], where there is one central scheduling node and each computation node collects statistics of its own usage. These data are then sent to the central node, and the scheduling algorithm adds the tasks from the queue to the queue of tasks of specific nodes according to the prediction of node utilization computed from the statistics. A system in which there is more than one scheduling node, is described in [10]. The scheduling nodes are independent machines and therefore have no information about schedules of others, so a statistical approach to node utilization is used. A two-level scheduling algorithm is described in [9]. The first level schedules the task to the specific "server", which is the leader of a set of close nodes. The "server" then schedules the task to a specific node. A very simple dynamic algorithm was also proposed in [23]. The idea is that the nodes are not differentiated, i.e. a node can be both a "worker" and a "server". The schedule is created in steps, and in each step several messages are sent that try to get information on the utilization of the neighbours of the node.

2.2 Weaknesses of the model

The model of the application (AM) describes the application in a very good way. However, this description is limited and does not reflect reality in all terms that we could imagine. There is, for example, a hidden prerequisite that all of the nodes know the code of the application and all the data that are needed as input for the application are available before the application is computed. Similarly, the output of the application is not targeted to a specific node, and the computation can finish on any node, which may be confusing in reality.

The computation system CS is also simplified. All of the properties of the network are merged in the two matrices $\mathbb{R}$ and $\mathbb{S}$. The values of the matrices do not take into account all of the possible properties of the network. In figure 3 the network contains one bottleneck. However, the properties of the network gained from independent measurements of the properties of the link do not indicate this, and communication may therefore be delayed against the plan of the schedule. However, this delay may or may not be critical for the subsequent computation and it is purpose of this paper to show how the communication delay may change the execution order of the schedule.

2.3 Related work

The problem of evaluating the correctness of the generated schedules has been studied extensively. Several tools have been presented, all of which try to help researchers to validate their algorithms. There are two main attitudes to the problem. Testing the algorithms on real platforms, and simulating the experiments. The problem with using real systems for testing is that there are very limited possible system architectures due to the limited hardware resources. There are, however, some systems that are focused on this type of testing. Grid'5000 [4] and PlanetLab [6] are two examples of platforms available for application testing. The results provided by these tools are very reliable, but the scalability of the network is limited. Another very important problem tightly coupled with task scheduling is that the number of existing applications is limited. Since only generated structures of non-existing applications are tested, real systems cannot be used. The same problem emerges when we talk about emulation tools.

Simulating experiments suits the task scheduling problem much better. This method is very widely used, though not all authors mention the system and the simulation method that they have used [16]. However, several systems became well known. GridSim [3] is a simulation tool focused on modeling the resources of the nodes of CS. The network layer is modeled using a simple discrete event simulation, and starts at the level of the third layer of ISO/OSI. The ALEA framework [12], an extension of GridSim, provides a tool for various GRID scheduling problems.

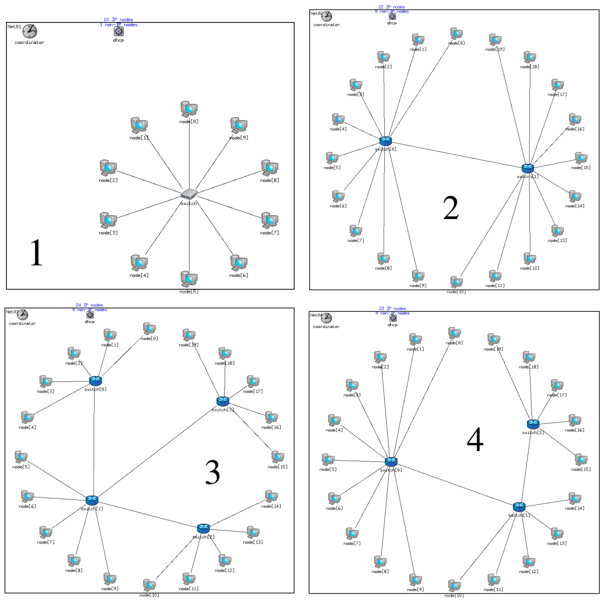


Figure 4: Topologies of four testing networks.

Another well-known simulator is SimGrid [5]. SimGrid has transformed from a tool for scheduling applications with a DAG structure to a system that is able to both simulate and emulate distributed algorithms. SimGrid uses a math model of the network, but the version 3 also introduces a hybrid system that allows the use of GTNetS [18] as a transport simulation tool.

3 Simulation tool

The purpose of this paper is to show the influence of network parameters on schedule execution. Hence the simulation tool has to offer a realistic simulation of the network, and we decided to use the OMNeT++ simulation tool [22]. OMNeT++ aims primarily at network simulations and is used as a core for many projects. There are also many extensions to OMNeT++, e.g. INet Framework is a set of OMNeT modules that simulate Internet devices. INet contains modules for both physical devices (routers, switches, hubs or access points for wireless networks) and protocols (TCP/IP protocol family, SCTP, OSPF or MPLS). Since we want to make the simulation of the network as realistic as possible, we chose OMNeT++ with INet Framework as a simulation core.

OMNeT++ itself offers no support for scheduling. As mentioned above, the applications that we use for testing the scheduling algorithms are randomly generated and have no real representation (i.e. code). The simulation of the execution of the schedule then consists only of sequences when the data is sent or received and when the nodes pretend to be working. In terms of simulation, they sleep for a specified amount

	R2
R1	10 GBit, 1000 m
R3	100 MBit, 100 m
R4	1 GBit, 10000 m

Table 1: Speeds and distances used in network 3.

	R2
R1	100 MBit, 100 ms
R3	10 MBit, 100 m

Table 2: Speeds and distances in network 4.

of time. This behavior is executed in the *TaskExecutor* module, which may be connected to two modules, the first for TCP communication and the second for UDP communication. The communication itself is then simulated by standard INet Framework. This involves packet collisions, routing, queuing of packets, etc.

4 Simulation results

We created four network topologies that were used for simulation. The topologies of the networks are shown in Figure 4, and the main properties of each of the networks are as follows.

Network 1 contains 10 nodes connected by a 1 GBit ethernet link to the central switch; the cable length is 10 meters.

Network 2 contains 2 groups of 10 nodes. The nodes are connected by 1 GBit ethernet to the router; the routers are connected by a 10 GBit point-to-point line with the delay corresponding to a distance of 10 km.

Network 3 contains 4 groups of 5 nodes. The nodes are connected by a 1 GBit ethernet to the router, and the routers are connected as shown in the Table 1.

Network 4 contains 1 group of 10 nodes and two groups of 5 nodes. The nodes are connected by 1 GBit ethernet to the router, and the routers are connected as shown in Table 2.

The communication delays are specified in time units (ms) or in distance units (m) – the real delay is then computed by the following equation:

$$delay = \frac{distance}{0.64c}. \tag{3}$$

Networks 1 to 4 were the computation systems for a set of 300 randomly generated applications. The method for generating them was copied from [20].

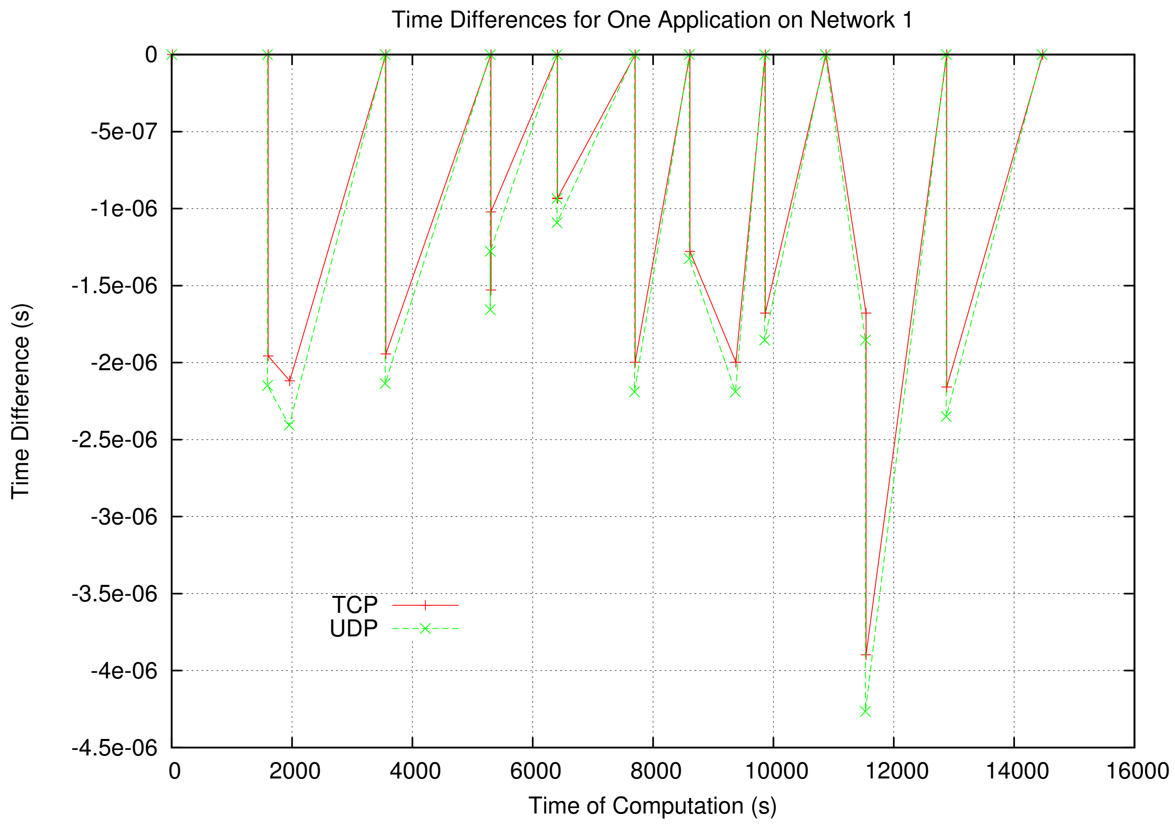


Figure 5: Differences between the expected and real start of tasks of one application. Network 1 as an execution platform.

A schedule for each network was created for each application. We used HEFT and CPOP as scheduling algorithms and TCP and UDP as the transport layer. In the end, we had a set of 4800 schedules. All of the schedules were simulated and the differences between the execution time of the scheduled task and the simulated task were recorded.

The results of the simulations showed that the differences between real execution time and expected execution time are not large. We had expected that these small differences would cumulate and grow, but the difference seems to be almost constant, see Figures 5, 6 and 7.

The structure of the network influences the execution of the schedule. It may not be a coincidence that the time differences are usually in the order of the startup delay. For example, the time differences in Network 1 were very slow, the maximum being about 10^{-6} s, which is the same order as the startup delay for Network 1 stored in $\mathbb{S}$. The differences in Network 4 were higher, 10^{-1} s, which is higher than the order of values in the startup delay matrix $\mathbb{S}$. Nevertheless, the value is much smaller than the total execution time of the computation.

We have also shown that the average length of the schedule was 10^3 s and the average difference caused by the network transport was -10^{-6} to -10^{-1} s. The difference is really small compared to the schedule length.

5 Conclusion

In this paper we have presented the problem of task scheduling. Since the problem is difficult, heuristics are used, and great progress has been achieved in this area. However, the models that are used as a standard input for most of the algorithms are only models, and may suffer from many simplifications.

We have focused here on the computation system, especially the networking subsection. We have shown that there are several points that may lead to misunderstandings about how the network may work. In order to show whether these points affect real world applications, we created a simulation tool which is able to simulate the execution of the schedule and the corresponding networking activity. The tool is based on OMNeT++, which is often used for network-based simulations.

We created a set of randomly generated applications, and schedules for them. We ran the simulations on four network topologies, and our results show that the communication caused some differences against the expected execution time. However, the differences were very small. For this specific set of appli-

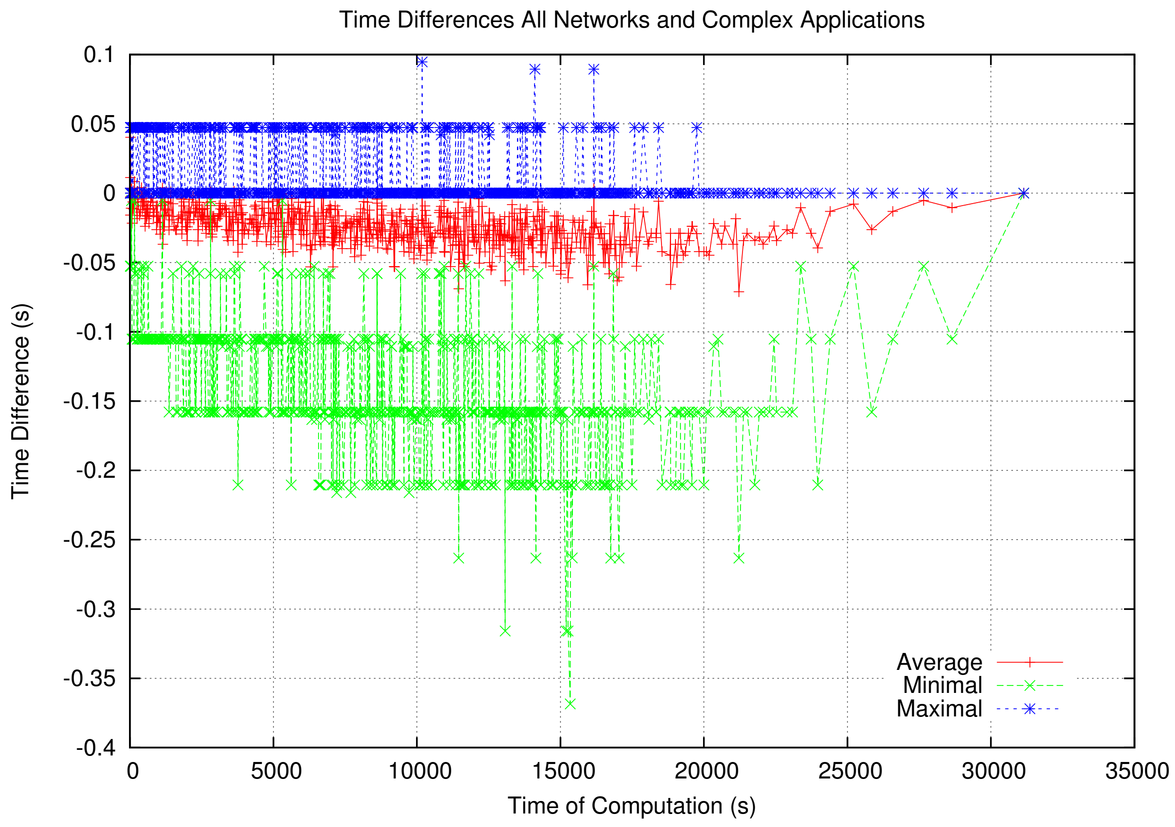


Figure 6: Differences between the expected and real start of applications with more than 50 tasks across all platforms.

cations and networks we may say that the differences are insignificant.

We also mentioned the idea that the time differences caused by the network, are in the order of startup delay. To make this idea bullet-proof we need to make more simulations on various types of networks and with various types of applications. Of course, the best way would be to execute real applications on real networks. Real networks have more problems than only collisions or delays. There may be some other traffic, and the bandwidth may therefore change during the computation. These and many other problems are still to be solved.

Acknowledgements

The research reported in this paper has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under research program MSM 6840770014 and by the Grant Agency of the Czech Technical University in Prague, grant No. SGS11/158/OHK3/3T/13.

References

[1] J. Barbosa, B. Moreira. Dynamic job scheduling on heterogeneous clusters. In *Parallel and Distributed Computing, 2009. ISPDC '09. Eighth International Symposium on*, pp. 3 –10. 2009.

[2] Tracy D. Braun, Howard Jay Siegel, Noah Beck, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* **61**(6):810 – 837, 2001.

[3] Rajkumar Buyya, Manzur Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience* **14**(13-15):1175–1220, 2002.

[4] F. Cappello, E. Caron, M. Dayde, et al. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pp. 99–106. IEEE Computer Society, Washington, DC, USA, 2005.

[5] Henri Casanova, Arnaud Legrand, Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE*

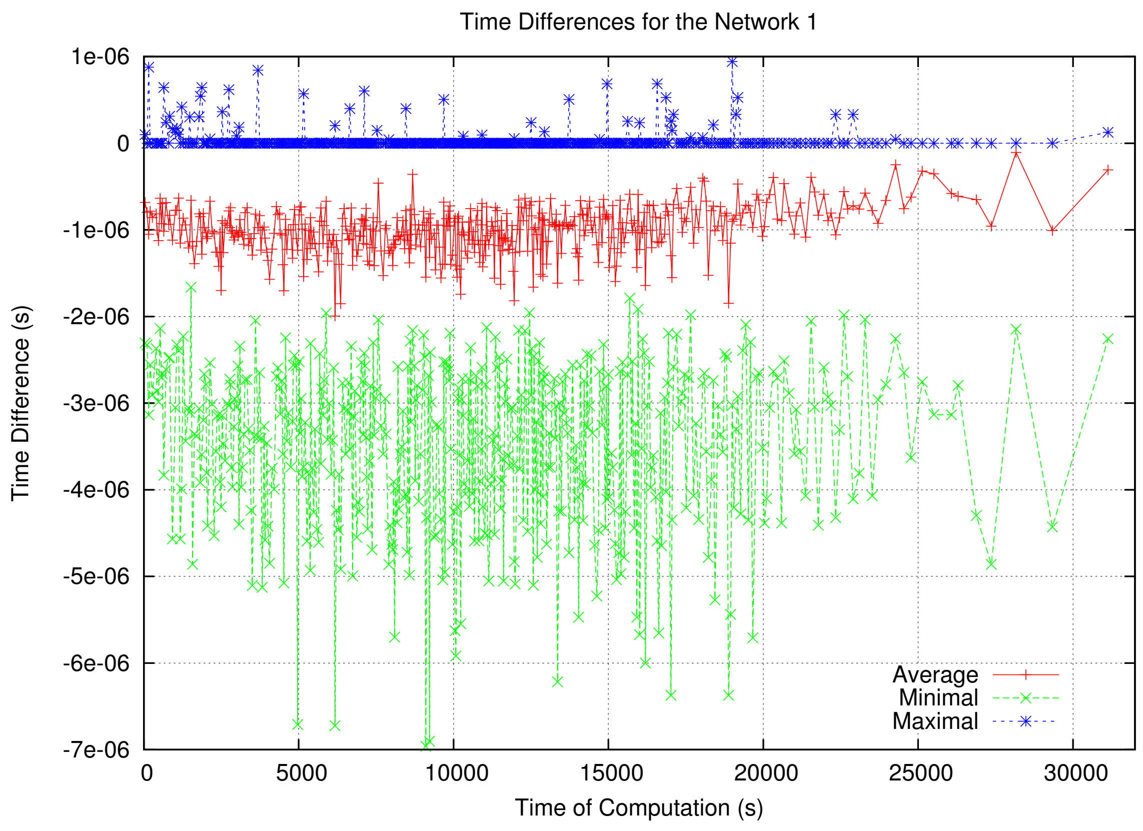


Figure 7: Average, minimal and maximal time difference between the expected and real start of all tasks of all applications executed on Network 1.

International Conference on Computer Modeling and Simulation. 2008.

[6] Brent Chun, David Culler, Timothy Roscoe, et al. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput Commun Rev* **33**:3–12, 2003.

[7] Michael R. Garey, David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[8] T. Hagras, J. Janecek. A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems. *Parallel Computing* **31**(7):653 – 670, 2005. Heterogeneous Computing.

[9] M A Iverson, F Özgüner. Hierarchical, competitive scheduling of multiple dags in a dynamic heterogeneous environment. *Distributed Systems Engineering* **6**(3):112, 1999.

[10] Michael Iverson, Fusun Ozguner. Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment. *Heterogeneous Computing Workshop* **0**:70, 1998.

[11] Jan Janeček, Peter Macejko, Tarek Morad Gomaa Hagras. Task scheduling for clustered heterogeneous systems. In M.H. Hamza (ed.), *IASTED International Conference - Parallel and Distributed Computing and Networks (PDCN 2009)*, pp. 115–120. 2009. ISBN: 978-0-88986-783-3, ISBN (CD): 978-0-88986-784-0.

[12] Dalibor Klusáček, Hana Rudová. Alea 2 – job scheduling simulator. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*. ICST, 2010.

[13] Yu kwong Kwok, Ishfaq Ahmad. Benchmarking the task graph scheduling algorithms. In *In Proc. IPPS/SPDP*, pp. 531–537. 1998.

[14] Claudia Leopold. *Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[15] Jiuxing Liu, B. Chandrasekaran, Jiesheng Wu, et al. Performance comparison of mpi implementations over infiniband, myrinet and quadrics. In *Supercomputing, 2003 ACM/IEEE Conference*, p. 58. 2003.

[16] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai. Towards yet another peer-to-peer simulator. In *Proceedings of The Fourth International Working Conference on Performance Modeling and Evaluation of Heterogeneous Networks (HET-NETs)*, pp. 37/1–37/10. Ilkley, UK, 2006.

[17] Xiao Qin, Hong Jiang. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. *Parallel Processing, International Conference on* **0**:0113, 2001.

[18] George F. Riley. Simulation of large scale networks ii: large-scale network simulations with gtnets. In *Proceedings of the 35th conference on Winter simulation: driving innovation*, WSC '03, pp. 676–684. Winter Simulation Conference, 2003.

[19] Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, New York, NY, USA, 2nd edn., 2001.

[20] H. Topcuoglu, S. Hariri, M.Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* **13**:260–274, 2002.

[21] J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences* **10**(3):384 – 393, 1975.

[22] András Varga, Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pp. 60:1–60:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2008.

[23] O. Votava, P. Macejko, J. Kubr, J. Janeček. Dynamic Local Scheduling of Multiple DAGs in a Distributed Heterogeneous Systems. In *Proceedings of the 2011 International Conference on Telecommunication Systems Management*, pp. 171–178. American Telecommunications Systems Management Association Inc., Dallas, TX, 2011.

[24] Henan Zhao, R. Sakellariou. Scheduling multiple dags onto heterogeneous systems. *Parallel and Distributed Processing Symposium, International* **0**:130, 2006.

[25] Albert Y. H. Zomaya (ed.). *Parallel and distributed computing handbook.* McGraw-Hill, Inc., New York, NY, USA, 1996.