Towards the Efficient Simulation of Telecommunication Systems in Heterogeneous Distributed Execution Environments

Gábor Lencse, István Derka and László Muka

***Abstract*—An introduction is given to the topics of the parallel and distributed simulation and of the modeling of telecommunications systems. Our practical modeling concept for simulation in heterogeneous execution environment is presented. Its logical topology is a star shaped network of homogeneous clusters. The load balancing and the coupling factor criteria are set up for building models of telecommunications systems so that the simulation may produce good speed-up in a heterogeneous distributed execution environment.**

A case study is given with the open source OMNeT++ discrete-event simulation system and its parallel CQN (closed queueing network) sample model executed by 64 CPU cores of four different types. Our criteria are heavily supported by the results of our experiments.

***Keywords*—discrete-event simulation, heterogeneous execution environment, MPI, OMNeT++, parallel and distributed simulation, telecommunication systems.**

I. Introduction

There is a rapid development in the technology of the telecommunication systems (both wired and wireless) but the global recession makes the service providers (or network owners) careful in their investments. This phenomenon brings the performance analysis of telecommunication systems into the focus of current research. *Performance analysis* [1] can give answers to questions, like: Where are the bottlenecks in my IP network? Can I guarantee certain QoS parameters in the following six months if the growth of the traffic will follow the current trends? What will be the traffic conditions if certain elements of the network will be replaced by given faster ones? Etc.

Event-driven discrete-event simulation [2] is a widely used method for the performance analysis of telecommunication systems. However, the precise simulation of large and complex networks may require an amount of computing power and memory that is often available only on a supercomputer. The usage of multiple computers instead is a natural idea, but the algorithm of the event-driven discrete-event simulation makes it a difficult problem (see details later). Parallel and distributed simulation has a large literature and we can make only a short survey of it focusing on the choice of the most appropriate method for telecommunication systems.

The clock speed of the CPUs of the computers was rising quite fast until 2002 when it reached 3GHz (see Intel Pentium 4), but it is rising much slowly since then and the increase of the computing power of CPUs comes from other sources such as parallelism. The current proliferation of the multi-core CPUs makes parallel and distributed simulation even more relevant research topic.

The global recession influences also the availability of computing resources for the purpose of simulation. It means not only that clusters of less expensive computers should be used instead of a supercomputer, but it may also mean that even the purchase of a new cluster of workstations with identical configuration is not possible, rather the already existing computers should be used, which are possibly differ from each other in certain parameters or perhaps also in their architecture.

The aim of this paper is to show how to exploit the computing power of multiple computers for simulation of telecommunication systems even if the computers are of different types. That is, how to use a *heterogeneous execution environment* for the simulation of telecommunication systems.

The rest of this paper is organized as follows. First, a brief summary of the synchronization methods of parallel and distributed simulation is given focusing on what method(s) can be used in the field of telecommunication systems. Second, our view of the most general model of the components of telecommunication systems is introduced. Third, our practical modeling concept for simulation in a heterogeneous execution environment is presented. Fourth, our test environment is described. Fifth, our experiments and results are presented and discussed. Sixth, the related work of other researchers is summarized. Finally, our conclusions are given.

II. Parallel and Distributed Simulation

A. Basic Concepts

1) Event-Driven Discrete-Event Simulation

The event-driven discrete-event simulation uses the so-called *Future Event Set* (FES, also called *event-list*) for storing the events that are to happen at a given *model time*. At the beginning of the simulation, some events are inserted into the FES. The general step of the simulation removes the 'first'

Manuscript received February 11, 2013. This research was supported by the TÁMOP-4.2.2/B-10/1-2010-0010 project and by the Széchenyi István University (15-3202-08).

Gábor Lencse, István Derka and László Muka are with the Department of Telecommunications, Széchenyi István University, 1 Egyetem tér, Győr, H-9026, Hungary,
(e-mail: {lencse,steve,muka}@sze.hu).

event (that is the event with the smallest time stamp) from the FES. Then it sets the model time (also called *virtual time*) according to the timestamp of the event and 'plays' the event. During this, some new event(s) may be produced and put into the FES. The algorithm can be formalized as follows:

```
initialize, insert certain events into the FES;
repeat
   remove the first event from the FES;
   CLOCK := the time of the event removed from
      the FES;
   process the event, during this insert some
      event(s) into the FES if necessary;
until (FES is empty) or (CLOCK > limit) or (for
      other reason, we must stop)
```

How can this algorithm be parallelized?

2) Possibilities for parallelization

The computing power of multiple processors can be utilized for the same simulation in a number of ways [3] using:

- replicated trials
- functional decomposition
- time-parallel approach
- space-parallel approach

If the model has the complexity that it does not fit into the memory of one computer and/or its execution time is unacceptable (e.g. weeks – depending on the situation) then the space-parallel approach should be used: the model is cut into segments, and the segments are assigned to processors that are executing them. This paper deals with this approach only.

3) The hardship of parallelization

Using one centralized FES for all the segments would be a bottleneck and the parallel simulation would not scale up well. Hence, the segments must use their own event sets. Thus, the first event is selected and executed independently in each segment: the segments (also called *logical processes*) have different *local virtual times*, which may result in causality errors. To avoid causality errors, the local virtual times of the segments must be *synchronized*.

B. Synchronization Methods

There are two well-known synchronization methods for parallel discrete-event simulation (PDES). They are described by [4].

1) Conservative Synchronization Method

The causality is ensured by the rule that only *safe* events can be processed. An event is safe if it is guaranteed that no events with a smaller timestamp may arrive from any other segments. The so-called *lookahead* is a very important factor concerning the possible speed-up. If the local virtual time of *segment A* is t_A, and the lookahead is τ then all the segments can be sure that no events will arrive from segment A with a timestamp smaller than $t_A+\tau$. The lookahead is the property of the simulation model. There may be a minimum time interval between events. Unfortunately, the models of telecommunication systems often use Poisson distribution for generating the timestamp of transactions, or exponential distribution for the length of the service time; they both result in zero lookahead. However a positive and sometimes even large lookahead may come from the delay of communications lines. Its value is proportional to the length of the lines and its relative value increases with the speed of the networks: the faster the networks are the more events may happen during the time that is necessary for a packet to travel for a given distance. This phenomenon makes the conservative synchronization method more usable now than it was decades before. This method is used in our paper.

2) Optimistic Synchronization Method

Any first events from the FES of a segment can be processed and the causality errors are detected when an event arrives to a segment with smaller timestamp than the local virtual time of the segment. This message is called *struggler*. A *rollback* is done: all the state changes happened in the system later than the timestamp of the struggler are reverted and anti-messages are sent for all the messages that were sent out with greater timestamp than that of the struggler.

There are two problems with this approach. The method does not scale up well: as the number of segments increases, the rollbacks cause larger and larger load and there will be no good speed-up. The other problem is more technical. This is the implementation of the rollback. It can be done by periodic state savings, but that may take too much time and may consume too much memory. Dynamic memory allocation may also cause problems so it should be done with the support of the simulation kernel.

As the optimistic synchronization method causes extra work for the writers of both the simulation kernels and the models it is not popular in practice. This method is not tested in our paper.

3) Statistical Synchronization Method

The Statistical Synchronization Method [5] is less well-known. It does not exchange individual messages between the segments but rather the statistical characteristics of the message flow. The method can produce excellent speed-up [6] but has a limited area of application [7]. However, it may be successfully used in the performance analysis of telecommunication systems therefore we plan to test it in a later research.

III. General Model of Telecommunication Systems

Even though the wired and wireless telecommunication systems can contain many types of elements, for the purpose of performance analysis, they can be modeled by a graph built of nodes and lines. In the simplest model, the nodes can be characterized by the number of packets they can forward in a second, and the lines can be described by their transmission speed (e.g. in Mbps) and their transmission delay. More sophisticated models may more or less precisely follow the different protocols that are used in the network.

The *queue* is a typical modeling element in discrete-event simulators. It can represent for example a router. A router stores the arriving packets in a buffer and forwards them into the right direction on the bases of the routing decision. The routing decision may take a fixed amount of time or if the routing table is ordered in the decreasing probability (approximated by their frequency in the past) of the routes then the popular routes are found faster and the less popular ones are found slower. Using a *queue* as a model of the router, service time can be fixed or may follow a certain distribution.

Telecommunication models typically use exponential distribution for modeling the service time. A queue can also represent a transmission line. Here, the service time is determined by the length of the packets divided by the speed of the line. Thus the distribution of the service time follows the distribution of the packet length. In addition to that, the delay of the line must be modeled too.

IV. Distributed Model Construction for Simulation in Heterogeneous Execution Environments

A. Our Concept of Heterogeneous Execution Environments

Theoretically, many levels of hierarchy and many kinds of topologies could be used. To be practical, we recommend a *logical topology* of two levels only: a *star shaped network* of *homogeneous cluster*s. This model is simple enough and can describe a typical *heterogeneous execution environment*. What is logically described as a homogeneous cluster, it can be physically, for example, a cluster of PCs with identical configuration interconnected by a switch or it may also be a chassis based computer build up by several main boards, etc. The main point is that a homogeneous cluster is built up by identical configuration elements especially concerning CPU type and speed as well as memory size and speed. The different homogeneous clusters are interconnected logically in a star shaped topology. The physical connection can be a switch or the topology may be different but our model considers it to be a star for simplicity.

Notes:

1. The above definition allows a homogeneous cluster to be built up by a single element only.
2. The elements of the homogeneous clusters can share the same type of executable code and they provide the same performance.

B. Constructing models to achieve a good speed-up

1) Load Balancing Criterion

The basic idea is very simple: all the CPUs (or CPU cores) should get a *fair share* from the execution of the simulation. A fair share is proportional to the computing power of the CPU *concerning the execution of the given simulation model*. (This is very important, because, for example, using different benchmark programs for the same set of computers one can get seriously different performance results.) Thus, for the fair division of a given simulation model among the CPUs, the CPUs should be benchmarked by the same type of simulation model that is to be executed by them (but smaller in size, of course). See more details later at the discussion of our test simulations.

Note that real life models cannot be arbitrarily cut into segments. Models usually can be cut along the boundaries of logical building blocks.

2) Lookahead or Coupling Factor Criterion

As the *lookahead* of telecommunication systems usually comes from the delay of transmission lines, the model should be cut into segments at the long distance lines.

At this point, some important questions come up:

- How many segments should be made?
- Is it worth using all the available computers?

To be able to answer these questions, some earlier research results should be recalled. The so-called *coupling factor* can be calculated from values that can be easily measured in a sequential simulation or directly in the execution environment [8]. The order of magnitude of the coupling factor gives a good hint if there is a chance for a good speed-up [9]. The available parallelism can be assessed using the following quantities (description is taken from [9]):

- *P performance* represents the number of events processed per second (*ev/sec*). P depends on the performance of the hardware and the amount of computation required for processing an event. P is independent of the size of the model.
- ***E*** *event density* is the number of events that occur per simulated second (*ev/simsec*). *E* depends on the model only and not on the hardware and software environment used to execute the model. *E* is determined by the size, the detail level and also the nature of the simulated system.
- *R relative speed* measures the simulation time advancement per second (*simsec/sec*). Note that $R = P/E$.
- ***L*** *lookahead* is measured in simulated seconds (*simsec*). When simulating telecommunication networks and using link delays as lookahead, *L* is typically in the *microsimsec–millisimsec* range.
- $\boldsymbol{\tau}$ *latency* (sec) is the latency of sending a message from one segment of the simulation model to another. This value is usually in the μs-ms range, and is largely determined by the hardware and software on which the simulation runs.
- $\boldsymbol{\lambda}$ *coupling factor* can be calculated as the ratio of *LE* and τP:

$$\lambda = \frac{L \cdot E}{\tau \cdot P} \tag{1}$$

The value of λ decreases with the number of segments. If we use N number of segments then:

$$\lambda_N = \frac{\lambda}{N} \tag{2}$$

Ref. [9] states that if λ is in the order of a couple times 100 or higher then we may expect good speed-up. It may be nearly linear even for higher number of segments if λ_N is also at least in the order of a couple of hundreds.

More details and our measured values will be given later when discussing the experiments.

V. Test Environment

A. Available Hardware Base

The following servers, workstations and PCs were available for our experiments at the Info-communications Laboratory of the Department of Telecommunications, Széchenyi István University.

1) Sun Server SunFire X4150

- Two Quad Core Intel Xeon 2.83GHz CPU
- 8GB DDR2 800MHz RAM
- Two near-line SAS 160GB HDD
- Two Intel 82571EB Gigabit Ethernet NIC
- Two Intel 80003ES2LAN Gigabit Ethernet NIC

Altogether it means a homogeneous cluster of 8 nodes.

2) Three LS21 Blades (in IBM BladeCenter E Chassis)

- Two Dual Core Opteron 280 2.4GHz CPU
- 4GB DDR2 667MHz RAM
- 73GB SCSI Ultra 320 HDD
- Broadcom NetXtreme BCM5704S Gb. Eth. NIC

Altogether it means a homogeneous cluster of 12 nodes.

3) Six Dell Precision 490 Workstations

- Two Intel Xeon 5140 Dual Core 2.33GHz CPU
- 4x1GB DDR2 533MHz RAM (quad channel)
- 80GB SATA HDD
- Four Intel 82571EB Gigabit Ethernet NIC
- Broadcom NetXtreme BCM5752 Gb. Eth. NIC

Altogether it means a homogeneous cluster of 24 nodes.

4) Ten AMD PCs

- AMD Athlon 64 X2 Dual Core 4200+ 2.2GHz CPU
- 2GB DDR2 667 MHz RAM
- Two 320 GB SATA HDD
- nVidia CK804 Gigabit Ethernet NIC

Altogether it means a homogeneous cluster of 20 nodes.

Switches for Interconnection

- 3Com Baseline Switch 2948 SFP Plus (3CBLSG48)
- Cisco Intelligent Gigabit Ethernet Switch Module, 4 ports (Part Number 32R1894) in the BladeCenter

B. Software Environment

1) Operating Systems

Linux was used on all the computers.

Sun Server and LS21 Blades: Ubuntu 12.04 LTS x86-64

Dell Precision 490 Workstations and AMD PCs: Debian Squeeze (x86_64)

2) Cluster Software

OpenMPI 1.6.2 (x86_64)

3) Discrete-event simulation software

The widely used, open source OMNeT++ 4.2.2 discrete-event simulation environment was chosen [10]. It supports the conservative synchronization method (the Null Message Algorithm) since 2003 [11].

VI. Experiments and Results

A. The Simulation Model

The Parallel CQN (Closed Queueing Network) sample simulation model of OMNeT++ was used for our experiments. We considered this model appropriate because it is built up by queues, lines and routing decision points, which ones are the typical elements of the models of telecommunication networks. The same model was used in [9]. The below description of the model is taken from there.

This model consists of M tandem queues where each tandem consists of a switch and k single-server queues with exponential service times (Fig. 1). The last queues are looped back to their switches. Each switch randomly chooses the first queue of one of the tandems as destination, using uniform distribution. The queues and switches are connected with links that have nonzero propagation delays. The OMNeT++ model for CQN wraps tandems into compound modules.

To run the model in parallel, the tandems should be assigned to different segments (Fig. 2). Lookahead is provided by delays on the marked links.

As for the parameters of the model, the preset values shipped with the model were used unless it is stated otherwise. Configuration B was chosen, the one that promised good speed-up.

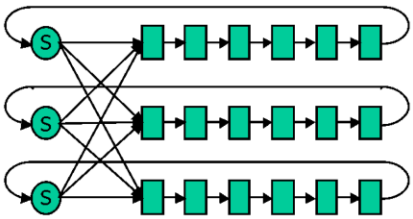

Fig. 1. M=3 Tandem Queues with k=6 Single Server Queues in Each Tandem Queue

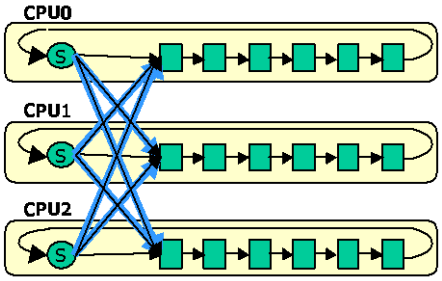

Fig. 2. Partitioning the CQN Model

B. Experimenting on a Homogeneous Cluster

As an introduction, series of experiments were performed on the Sun server. The main parameters of the CQN model were set to the same as in [9]: M=24 tandem queues, k=50 queues in each tandem queue, exponential service time of the queues with expected value of 10 seconds, the delay between the tandem queues L=100 seconds. A single processor simulation was executed to measure the E and P variables necessary for the calculation of λ. (OMNeT++ directly displays these values both in its GUI and command line environment.) The communication latency between the cores was measured by the **`pingpong`** OpenMPI test program. We got 10^{-6} seconds for the value of latency.

The value of λ was calculated as follows:

$$\lambda = \frac{100\text{simsec} \cdot 156\text{ev/simsec}}{10^{-6}\text{sec} \cdot 468192\text{ev/sec}} \approx 3000 \qquad (3)$$

Series of experiments were performed executing the CQN model divided into N segments running each segment on its own CPU core. The values of N were: 1, 2, 4, and 8. N=1 means that there was only a single segment. The length of the simulation was set as 10^6 seconds in model time (*simsec*) in order to have a reasonably long execution time. (All the experiments

were performed 11 times and the average and the standard deviation of the execution time were calculated. This applies to all the later series of experiments.) Table I shows the results. They show an excellent speed-up.

TABLE I
EXECUTION TIME AND SPEED-UP IN THE FUNCTION OF THE NUMBER OF SEGMENTS WITH **L=100S** LOOKAHEAD – SUN SERVER ONLY

Number of segments	1	2	4	8
Average execution time (sec)	332,50	183,18	91,33	48,70
Std. dev. of the execution time	2,75	4,41	3,85	1,96
Speed-up	-	1,82	3,64	6,83
Relative speed-up	-	0,91	0,91	0,85

To demonstrate a case with a poor speed-up, the lookahead was decreased from 100s to 1s (in model time). This change results in approximately 100 times smaller lambda (but not exactly, because the values of the lookahead influenced also the event density in the model):

The value of λ was calculated as follows:

$$\lambda = \frac{1\text{simsec} \cdot 160\text{ev/simsec}}{10^{-1}\text{sec} \cdot 453739\text{ev/sec}} \approx 350 \quad (4)$$

This value still anticipates a good speed-up for two segments, however the situation is different for 8 segments:

$$\lambda = \frac{\lambda}{8} = \frac{350}{8} \approx 44 \quad (5)$$

The results in Table II show to support the theory presented in [8] and [9]. The simulation produced longer execution time using 8 cores than using 4 cores.

C. *Experimenting with Load Balancing*

The simplest inhomogeneous execution environment contains two homogeneous "clusters" – each of which is actually a single computer (or CPU core). Our test system had one core from the Sun server and one core from the IBM BladeCenter interconnected by a 3Com Baseline 2948SFP Plus switch. The value of τ between the two computers was 25µs. The simulation model was the same as before with L=100s lookahead. First, the computers (the CPU cores) were benchmarked with the simulation model (performed as sequential simulation). Table III shows the results (also for the two other CPU types which were used later on).

Next, the following series of experiments were performed: the CQN model was cut into two segments: N and 24-N tandem queues were put into the segment executed by the Blade and the Sun, respectively, where N took the values form 1 to 23. For the purpose of comparison, the N=0 and N=24 cases were also executed, that is the simulation was executed sequentially by the Sun and the Blade, respectively. Fig. 3 shows the results.

The measurement results show that the partitioning was the best when 9 and 15 tandem queues were put into the segments executed by the Blade and the Sun servers, respectively, but the results for 8 and 16 tandem queues are also very close. The exactly *performance proportional partitioning* would result in the assignment of 8.1 and 15.9 tandem queues. The computed "optimal partitioning" is quite close to the results of the experiments.

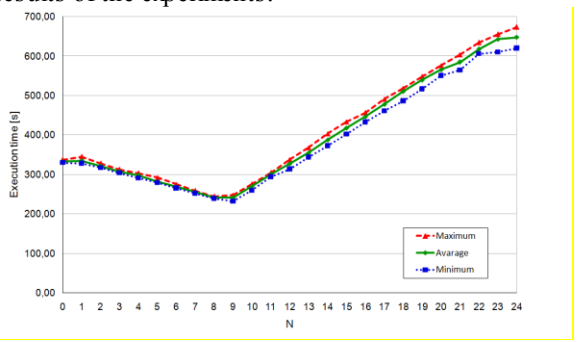

Fig. 3. Execution Time of the CQN Model in the Function of the Partitioning: there were N and 24-N tandem queues put into the segment executed by the Blade and the Sun servers, respectively.

TABLE II
EXECUTION TIME AND SPEED-UP IN THE FUNCTION OF THE NUMBER OF SEGMENTS WITH **L=1S** LOOKAHEAD – SUN SERVER ONLY

Number of segments	1	2	4	8
Average execution time (sec)	352.22	220.16	190.15	242.54
Std. dev. of the execution time	2.13	1.47	1.34	2.24
Speed-up	-	1.60	1.85	1.45
Relative speed-up	-	0.80	0.46	0.18

TABLE III
THE PERFORMANCE OF THE SUN, IBM, DELL AND AMD COMPUTERS (EVENTS/SECOND)

Core Type	Sun	IBM	Dell	AMD
Average	468 192	238 174	373 787	235 861
Std. dev.	5 581	6 049	4 908	2 726

D. *Looking for the Best Available Speed-Up*

A large heterogeneous system was set up including 8 Sun cores, 12 Blade cores, 24 Dell cores and 20 AMD cores that is altogether 64 cores from four types. Fig. 4 shows the topology of the system. The number of the tandem queues was increased in the CQN model from 24 to 480 to be able to utilize all the CPU cores. The value of the lookahead was increased from 100s to 1000s to ensure a large enough value for λ_{64}.

Before experimenting with the heterogeneous system, the CQN model was executed by all the possible maximum size homogeneous clusters in order to have references for the calculation of the speed-up of the heterogeneous system. It means that the 480 tandems were divided into 8, 12, 24 and 20 partitions and they were executed by the homogeneous Sun, IBM, Dell and AMD clusters, respectively. The results are shown in table IV.

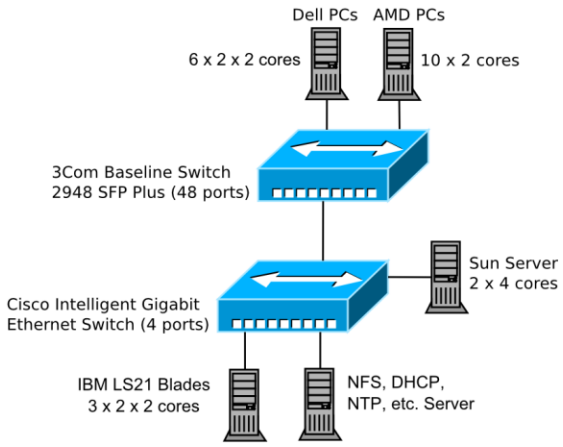

Fig. 4. The Heterogeneous Distributed Execution Environment for Testing

TABLE IV
THE EXECUTION TIME OF THE 480 TANDEM QUEUES BY DIFFERENT HOMOGENEOUS CLUSTERS

Cluster type	Sun	IBM	Dell	AMD
Number of cores	8	12	24	20
Average execution time	1 170.83	1 241.19	359.96	658.20
Std. dev. of exec. time	52.52	20.91	4.82	4.95

For experimenting with the heterogeneous system, the tandem queues were divided into partitions proportionally with the performance of the cores. As the model was formally "changed", the CPU cores could be benchmarked again, but as the nature of the model remained the same, the performance values of the CQN model with 24 tandem queues were used. Note that it must be done in the same way when simulating real life systems: the model for benchmarking the CPUs must be much smaller in size than the real model to be executed otherwise the benchmarking would take too much time.

Theoretically, if the number of the CPU core types is denoted by NCT, the number and the performance of CPU cores available from core type i are denoted by N_i and P_i, respectively then the n_i number of tandems to put into a segment executed by a core from type i should be:

$$n_i = \frac{480 \cdot P_i}{\sum_{j=1}^{NCT} N_i} \approx 2.364 \cdot 10^{-5} \cdot P_i \qquad (6)$$

However, the number of the tandem queues per segments must be an integer, thus the division of the tandems could not be fully precise, some "roundings" were done manually and there were differences made even between the load of the cores from the same core type (16 AMD cores had 5 tandem queues each but 4 AMD cores had 6 tandem queues each). Table V shows the division of the tandems among the cores.

The results of the execution of the simulation by the 64 cores of the heterogeneous system are shown in Table VI. They show different but significant speed up compared to any of the homogeneous clusters as references, thus we can conclude that it is worth using the heterogeneous system instead of any of the homogeneous clusters.

TABLE V
THE DIVISION OF THE 480 TANDEM QUEUES AMONG THE CORES

Core type	P_i	N_i	n_i	no. of cores	tandems /core	cumulated tandems
Sun	468 192	8	11.08	8	11	88
IBM	238 174	12	5.63	12	6	72
Dell	373 787	24	8.84	24	9	216
AMD	235 861	20	5.58	16	5	80
				4	6	24
			Number of tandems in the whole system:			480

TABLE VI
THE EXECUTION TIME OF THE 480 TANDEM QUEUES BY THE HETEROGENEOUS CLUSTER – THE SPEED-UP CALCULATED AGAINST THE DIFFERENT HOMOGENEOUS CLUSTERS

Execution time (s)		Speed-up against			
average	std. dev.	Sun	IBM	Dell	AMD
197.86	9.06	5.92	6.27	1.82	3.33

E. *Further Experiments*

We have carried out further experiments after the submission of this paper. The results of those experiments can be found in [12].

F. *Discussion of the Validity of the Results*

Our model was built up by queues, the typical elements used for modeling telecommunication networks. However the values of the parameters are questionable. In some of our experiments, the lookahead was chosen as 1000 seconds while the expected value of the service time of the elementary queues of the tandem queues was 10 seconds. That is the lookahead between the segments was chosen 100 times more than the typical service time of a packet. Is it a realistic assumption for a telecommunication network?

Now let us examine it in a short case study. The majority of the traffic of modern telecommunication networks comes from computer networks. Let us consider a system that interconnects three computer networks each of which resides in one of the following capitals: Budapest, Prague and Rome. If Gigabit Ethernet is used in the computer networks, the service time of the smallest (64bytes long) packets and the largest (1518bytes long) packets are 5.12×10^{-7} seconds and 1.2144×10^{-5} seconds, respectively. Even the shortest distance between Budapest and Prague is longer than 500km. If optical communication is used, the delay between these two sites is about 2.5×10^{-3} seconds, which is at least two orders of magnitude higher than the service time of any packet. Therefore, the model of the three sites can be efficiently executed in parallel by three processors as the delays of the long distance lines ensure the necessary values of the lookahead.

Note that the situation would be different in the case of lower data speed and/or shorter distances.

Thus the lookahead values used in our model can be realistic in some real life simulations with high enough data speed and long enough distances.

VII. Related Work

To find appropriate methods for the performance improvement of the execution of parallel and distributed simulation in heterogeneous cluster environment, there has been made a lot of relevant researches. The approaches which are related to the results introduced by the present paper can be divided into three groups.

The first group of approaches uses performance prediction of simulation in order to support planning and improving the efficiency. The method described in [13] interprets the efficiency improvement task as a scheduling and assignment problem and tries to solve it by using linear programming approach. The most important result of this work is the description of the theoretical limit of the execution improvement. Ref. [14] introduces an event-trace-based performance prediction tool which can be used for the practical planning of the simulation execution as a critical path analysis instrument. The use of methods described in [13] and [14] is strongly limited by the necessity of the application of special tools since they are tool based approaches. The method described in [15] is based on an analytical performance model of the conservative null message-based synchronization method. It may help to improve the partitioning of parallel simulation but it is not really convenient for a practical use.

The second way of approaches concentrates on the discovery of resources in the network that are suitable for the execution. Then, the results of the discovery process can be used in planning and scheduling of the execution process. For example, ref. [16] introduces methods of resource discovery and categorization for large-scale grid networks.

The third group of related approaches focuses on the process of autonomous load balancing in the network. Papers [17] and [18] introduce this approach well. Paper [17] proposes a bargaining based self-adaptive auction method for the load balancing while paper [18] looks at load balancing as the process of distributed negotiations among autonomous self-interested network computing peers. According to these approaches, there is no need for preliminary knowledge about the resources of the network, which is an advantage but the loss of control in the execution process may lead to efficiency decrease, which is a disadvantage.

VIII. Conclusion

A simple and practical modeling concept was proposed for simulation in heterogeneous execution environments. Its logical topology is a star shaped network of homogeneous clusters. The load balancing criterion and the coupling factor criterion were set up for building models of telecommunication systems so that the simulation may produce good speed-up in a heterogeneous distributed execution environment. Both criteria use the results of preliminary benchmarking the processors of the execution environment with a smaller version of the given simulation model to be executed.

The usability of the criteria was demonstrated on a heterogeneous execution environment built up by 64 CPU cores of four different types. Different test scenarios were conducted: good or poor speed-up was demonstrated depending on the size of the lookahead, the load balancing was justified by the measurement of the execution time of a model in the function of its partitioning and finally, it was demonstrated that a heterogeneous execution environment can overperform all its building homogeneous parts.

We conclude that it is a promising idea to use a heterogeneous execution environment for the simulation of telecommunications systems and our criteria may help to achieve a good speed-up.

References

[1] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, New York, NY, 1991.

[2] J. Banks, J. S. Carson, B. L. Nelson, *Discrete-Event System Simulation*. Prentice Hall, Upper Saddle River, NJ, 1996.

[3] J. Liu, "Parallel Discrete-Event Simulation" in *Wiley Encyclopedia of Operations Research and Management Science*, 2010. John Wiley and Sons.

[4] R. M. Fujimoto, "Parallel Discrete Event Simulation", *Communications of the ACM*, vol. 33, (1990.) no 10, pp. 31-53

[5] Gy. Pongor, "Statistical Synchronization: a Different Approach of Parallel Discrete Event Simulation", *Proceedings of the 1992 European Simulation Symposium (ESS'92)* (Dresden, Germany. Nov. 5-8) SCS Europe, pp. 125-129.

[6] G. Lencse, "Efficient Parallel Simulation with the Statistical Synchronization Method", *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)* (San Diego, CA. Jan. 11-14). SCS International, pp. 3-8.

[7] G. Lencse, "Applicability Criteria of the Statistical Synchronization Method", *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation (CNDS'99)*, San Francisco, CA. (1999. Jan. 17-20) SCS International, pp. 159-164.

[8] A. Varga, Y. A. Sekercioglu and G. K. Egan. "A practical efficiency criterion for the null message algorithm", *Proceedings of the European Simulation Symposium (ESS 2003)*, (Oct. 26-29, 2003, Delft, The Netherlands) SCS International, pp. 81-92.

[9] G. Lencse and A. Varga, "Performance Prediction of Conservative Parallel Discrete Event Simulation", *Proceedings of the 2010 Industrial Simulation Conference (ISC'2010)* (Budapest, Hungary, 2010. June 7-9.) EUROSIS-ETI, pp. 214-219.

[10] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment", *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, (Marseille, France, March 3-7, 2008) pp. 1-10.

[11] A. Varga and A. Y. Sekercioglu, "Parallel Simulation Made Easy with OMNeT++", *Proceedings of the 15th European Simulation Symposium (ESS 2003)*, (Oct. 26-29, 2003, Delft, The Netherlands) SCS International, pp. 493-499.

[12] G. Lencse and I. Derka "Testing the Speed-up of Parallel Discrete Event Simulation in Heterogeneous Execution Environments", unpublished.

[13] G. Kunz, S. Tenbusch, J. Gross, and K. Wehrle, "Predicting Runtime Performance Bounds of Expanded Parallel Discrete Event Simulations", In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. pp. 359-368.

[14] Z. Juhasz, S. Turner, K. Kuntner and M. Gerzson, "A Performance Analyser and Prediction Tool for Parallel Discrete Event Simulation", *International Journal of Simulation*, vol. 4, no. 1. (May 2003.) pp. 7-22.

[15] C. H. Li, A. J. Park and E. Schenfeld, "Analytical Performance Modeling for Null Message-Based Parallel Discrete Event Simulation", In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. pp. 349-358.

[16] K. Karaoglanoglou and H. Karatza, "Directing Requests and Acquiring Knowledge in a Large-Scale Grid System", *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2012)*, (Genoa, Italy, July 8-11), pp. 609-617.

[17] H. Zhao, and X. Li, "Efficient grid task-bundle allocation using bargaining based self-adaptive auction", In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. (2009, May). pp. 4-11.

[18] H. Zhao, and X. Li, "Hypergraph-based task-bundle scheduling towards efficiency and fairness in heterogeneous distributed systems", In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. (Atlanta, Georgia, USA, April 19-23), pp. 1-12.
