Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 119 (2017) 139–146

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Young Scientists Conference in HPC and Simulation, YSC 2017, 1-3 November 2017, Kotka, Finland

Adaptive load balancing of distributed multi-agent simulations on heterogeneous computational infrastructures

Oksana Severiukhina*, Pavel A. Smirnov, Klavdiya Bochenina, Denis Nasonov and Nikolay Butakov

ITMO University, Saint Petersburg, Russia

Abstract

Simulation of the agent-based model has several problems related to scalability, the accuracy of reproduction of motion. The increase in the number of agents leads to additional computations and hence the program run time also increases. This problem can be solved using distributed simulation and distributed computational environments such as clusters and supercomputers. The model objects must be divided into different processes and calculations to be able to be executed in parallel. This paper presents the research on an algorithm to balancing of computational load. The algorithm is based on a well-known genetic algorithm and performs optimization of matching between the model structure formed by multiple interconnected executable blocks of a distributed programming implementation of the model and network structure of the computational environment. Efficient placement of the model graph's nodes to heterogeneous computational ones leads to improvement in overall performance of the simulation and reducing of execution time.

Peer-review under responsibility of the scientific committee of the 6th International Young Scientist conference in HPC and Simulation

Keywords: agent-based modeling; distributed computing; load-balancing

1. Introduction

Modeling of large-scale and complex systems is an important challenge. Such simulations can find applications in different fields. Reproducing natural crowd movement is one of them. One of the wide accepted methods of modeling and studying various social processes and phenomena, such as the movement of people, transport, etc., are agent

* Corresponding author. *E-mail address:* oseveryukhina@gmail.com

1877-0509
Peer-review under responsibility of the scientific committee of the 6th International Young Scientist conference in HPC and Simulation
10.1016/j.procs.2017.11.170

models. One of the traits which is common for such simulations is a requirement for the high amount of computational resources. These resources are often spent on parameters investigation (for example, grid search) and repetition in order to accumulate enough statistical data. Each instance of such simulation consists of multiple blocks that can be run on different computational nodes and make use of distributed programming frameworks (for example, written using MPI).

Meanwhile, the structure of modern computational environments such as clusters and supercomputers may assume heterogeneity in network connections. Often their nodes are connected with one of the well-known typical topologies such as fat tree [1], butterfly [2], dragonfly [3] and others. The most widely accepted of them is a fat tree. This topology has a complex hierarchical structure with several levels that differ by bandwidth and point-to-point throughput for nodes using them in different segments (e.g., leafs of the fat tree). The topology of the computational network may also affect the overall computations performance, especially in case of federated infrastructures. For example, simulations may be scale by several remote clusters connected via the internet. However, additional network-specific information is required for better clusters distribution and agents' exchange. Information about bandwidths from different clusters of racks may be used for better agents' distribution and a decrease of non-optimal communication overheads. Authors [4] use genetic algorithm for workflow scheduling in case of heterogeneous distributed environment.

Individual instances of such models for efficient execution may be placed on nodes from the same network segments to avoid overheads implied by transferring between different segments. But often there are too many instances and they cannot be placed only in one segment and must use resources from few ones. Also, if there are other users who use this computational environment such situation can happen due to their workload.

From the other side, the structure of the application implementing such a model depends on the configuration of this model. Nowadays, multi-agent modeling is used for the different scientific area: pedestrian movement transport modeling, information distribution, modeling biological processes and so on. Each of the areas has its own characteristics, which must be taken into account during modeling this process. One of the important parameters is the environment in which the agents are located. A particular case of such an environment is a rectangular area. This can occur when modeling traffic on the highway, blood cells in the body [5] or movement of the crowd [6].

This can be illustrated with the following example. In Figure 2, one can see a model for simulation of agent movements in the city of Saint-Petersburg. In the model, individual districts and municipalities are connected between themselves and form a graph where edges are weighted. Their weights express the expectation of volume of agents that will be transferred between districts. A district itself can be viewed as a cluster of agents. However, it can be divided further into subclusters to obtain, even more, fine-grained structure, in this work we limit splitting of the model to district level only.

The important thing is that both graphs – topology of the network in a computational environment and cluster-based structure of the model being computed – introduce the notion of nodes proximity and require carefully adjusted matching of one graph to another to be efficiently executed. Neighborhood nodes in the graph of the model with high volumes of transferred agents should be placed on the topology graph on computational nodes with high bandwidth while weakly connected can be moved to different regions in the network topology. These conditions lead to an optimization problem of two graphs matching.

This paper contributes with:

1. An algorithm of matching model structure to network topology based on genetic algorithm.
2. Experimental study of the proposed algorithm's efficiency.

2. Related works

Wang and Lees distinguish several approaches to organizing a distributed computing scheme for the agent model of pedestrian movement [7]: centralization of modeling space; duplication of modeling space; division of space into zones, including agents located in this zone; independent separation of space and agents.

In the first case, one process is responsible for the state of the modeling space; agents are distributed among the remaining processes. This approach has a significant drawback: when modeling large-scale territories with a large number of agents, the bottleneck problem arises, associated with a large number of calls to the process responsible for the state of the modeling space. The second approach solves this problem by creating additional copies of the modeling

space, but it has many limitations related to the fact that with dynamically changing territory it takes time to synchronize space data on different processors. In the third approach, space is divided into separate zones. Each process has data about its zone, agents located on its territory, and the data of processes with which it interacts. For example, this is applicable for multiscale agent-based simulation. In work [8] there are two level of simulation: building and district, and agents move between them. The latter approach is based on dividing the modeling domain and agents into two different process groups. In this approach, it is more difficult to organize data consistency, in addition, more processes are required to achieve good model performance.

Parallel scheme for simulating mob motion using MPI technology was proposed in [9]. The simulation area is divided into separate blocks. Each block contains adjacent zone. Agents in this zone effect on agents in related zones.

The work [10] describes the implementation of distributed multi-agent modeling, using the model of social forces. The work uses the control and work processes: each worker models a separate area and returns the actual data to the master process. Processes exchange data when agents move from one area to another, as well as when agents are located in the adjacent zones. The proposed model was implemented in C language using MPI technology. The more complicated approach was proposed in work [11]. Authors use cluster-based portioning for the division of agents on different groups.

In contrary to the existing works, our study investigates the organization of load balancing by taking into account both structure of the network formed by the clusters in the model itself and heterogeneous network structure of the computational environment.

3. Algorithm

This section contains the description of the algorithm proposed in the study.

The simulation itself consists of several steps that take place before the actual iterations of modeling and between them.

Figure 1 shows the general modeling scheme; it consists of 3 separate phases:

- Initialization phase: definition of basic parameters at the beginning of simulation;
- Regular phase: update the parameters for each iteration during the simulation;
- Final phase: saving the simulation parameters and terminating the program.

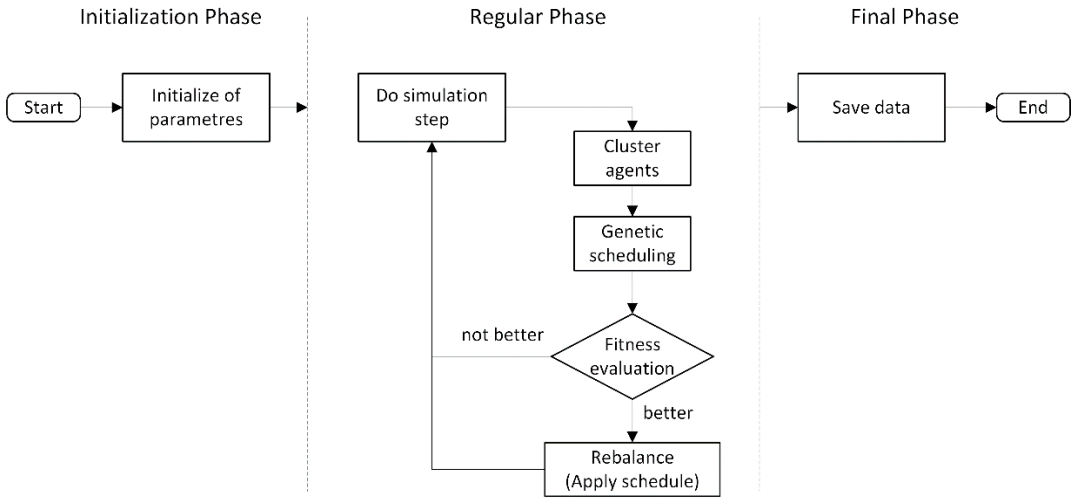

Fig. 1. Simulation scheme.

Initialization of parameters: There are two types of parameters. The first group includes parameters of objects of modeling: the number of agents, their initial position, parameters of the motion and the simulation area. The second group defines characteristics of computers: the number of processes and their location.

Cluster agents: During simulation agents interact with each other. Hence their positions depend on position and movement other agents, which are located near. The modeling area was divided into parts, which are determined by districts and municipalities of Saint-Petersburg (Fig. 2). The relationships between clusters are determined as number of agents transferring between districts.

Exchange of agents: During the simulation, agents move on modeling area and go to the zone of influence of another process. It may happen in the case of reaching the boundary of the zone or in the cases of recalculating the boundaries of the modeling areas belonging to the process. This block exchanges agents between different processes. In the proposed optimization algorithm, only expectation of an amount of agents that may be transferred between cluster is used.

The proposed algorithm (see block Genetic scheduling) can be used in dynamic conditions thus reevaluating the old matching and forming a new one if necessary in certain intervals of time of simulation. But in this work, we only consider the static case of initial cluster placement optimization and leaving the careful investigation of the dynamic case to future work.

The formal problem statement can be formulated as follows. Two graphs are assumed to be known $G_c = (V_c, E_c)$ - network graph of computational environment, $G_s = (V_s, E_s)$ - structure graph of the simulation. The goal function is:

$$y = argmin_{m \in M} f(m),$$

$$f = \max_{s=1..n} \max\left(t_s^{comp} + t_s^{send}, t_s^{receive}\right),$$

$$t_s^{comp} = E[a_s] * perf,$$

$$t_s^{send} = \max_{nb \in neigbors} E[a_s^{nb}] * E[a_s] * thr(s, nb),$$

$$t_s^{receive} = \max_{nb \in neigbors} E[a_{nb}^{s}] * E[a_{nb}] * thr(nb, s),$$

where $m = \{< v_c^i, v_s^j >: v_c^i \in V_c, v_s^j \in V_s\}$ – mapping of nodes to executable blocks of the simulation, a_s- amount of agents on executable block s, $E[a_s]$ – math expectation of the amount. $perf$ – performance of the node this block is placed on (assumed the same for all nodes in the experimental study), $neighbors$ – set of all nodes that should receive data from s or send data to s, a_s^{nb} - amount of agents has to be transferred from s to nb, $thr(s, nb)$ – the function that estimates amortized time of transferring between nodes holding executable blocks s and nb taking into account structure of the network.

Using this formula, the genetic algorithm is used for cluster-to-node assignments according to nodes performance and bandwidths in the heterogeneous computational network. The fitness function of the algorithm is composed of the following components: send-, receive- and calculation-time. Send and receive times reflect communication overheads for sending/receiving bordering agents to/from other clusters. Communication times send, receive depend on the amount of exchanging agents, their size, and bandwidths between particular nodes. Calculation time depends on final an amount of agents, which belongs to the cluster after send/receive operations finished. As a result, a schedule (clusters-to-node assignments) makes possible to calculate processing times for all the clusters. While clusters are processed in parallel the longest processing time will be a particular simulation step execution time.

We use a standard genetic algorithm applied to scheduling problems in many works before, for example [12][13], with modifications of fitness (calculated according to the formulas presented above), chromosome and mutation. The chromosome is an ordered set of pairs cluster id (executable block) – node id. All chromosomes have a fixed length equal to count of clusters. The mutation operators randomly choose a pair and change assigned node id to a randomly chosen one. The algorithm is initialized with a population of randomly generated solutions. The crossover operator is two-point crossover [14].

4. Experimental study

To test the efficiency of the proposed algorithm, we use parameters of infrastructure typical for small-sized cluster*. We suppose that communication environment of a testbed is organized as a fat tree and contains 3 root switches and 10 edge switches. Each edge switch is connected to one blade enclosure. In turn, each enclosure contains 10 nodes with 8 processors per node. Thus, each edge switch is connected to 80 processors, and the testbed includes 30 nodes and 240 processors. Speeds of data transfer are 2 Mb/s for 4-hops transfer (exploiting a root router), 100 Mb/s for 2-hop transfer (edge router), and 512 Mb/s inside an enclosure. A size of one agent is equal to 3 KB.

As distributed infrastructures are usually used in a non-dedicated mode (see, e.g. [15]), in the experiments we consider a situation of partial availability of a resource pool. We consider different levels of availability (% of processors busy by third-party tasks). Also, we use an assumption that free processors are distributed uniformly between nodes. The parameters of environment for different availability levels are summarized in Table 1.

Table 1. Parameters of environment for different availability levels.

% of busy processors	Free processors count	Free processors per node (format — nodes count: free processors)
60	96	24: 3, 6: 4
65	84	24: 3, 6: 2
70	72	12: 3, 18: 2
75	60	30: 2
80	48	18: 2, 12: 1
85	36	6: 2, 24: 1

For each configuration from Table 1, a matrix H [p×p] was generated where Hij is a number of hops between the ith and jth processor (1 for processors within one envelope, 2 for those within the same edge switch, and 4 otherwise). This matrix is using as an input data for the simulator and can be easily converted to a matrix with communication speeds between processors (using speeds of data transfer given above).

We compare the proposed algorithm with well-known round-robin scheduling algorithm [16] which switches between different enclosures and fills them sequentially, e.g., all free cores in one enclosure, then all enclosures under the same edge switches then all edge switches under the root switch. The modified genetic algorithm used the following parameters: population size – 400; count of generations – 50; crossover probability – 0.3; mutation probability – 0.7.

We test these two algorithms on the agents' mobility on Saint-Petersburg city map. As the agent clusters, the city municipalities were taken with the appropriate population rate. For studying the algorithm efficiency, we suppose the municipalities interactions as follows:

$$e_{a_s,a_s^{nb}} = r * \min(a_s, a_s^{nb}),$$

where e_{s_i,s_j} - amount of agents that goes from cluster s_i to cluster s_j each iteration, r = [0,1]- intensive coefficient that shows what part of agents are moving from one cluster to another. The map of clusters is presented in Fig. 2, where Fig.2.a presents the map of all municipalities, while Fig.2.b shows formed edges (e_{s_i,s_j}) for neighborhoods. This use case is taken for its cluster heterogeneity (performance load) that corresponds to the real city population distribution.

* http://hpc.msu.ru/?q=node/60

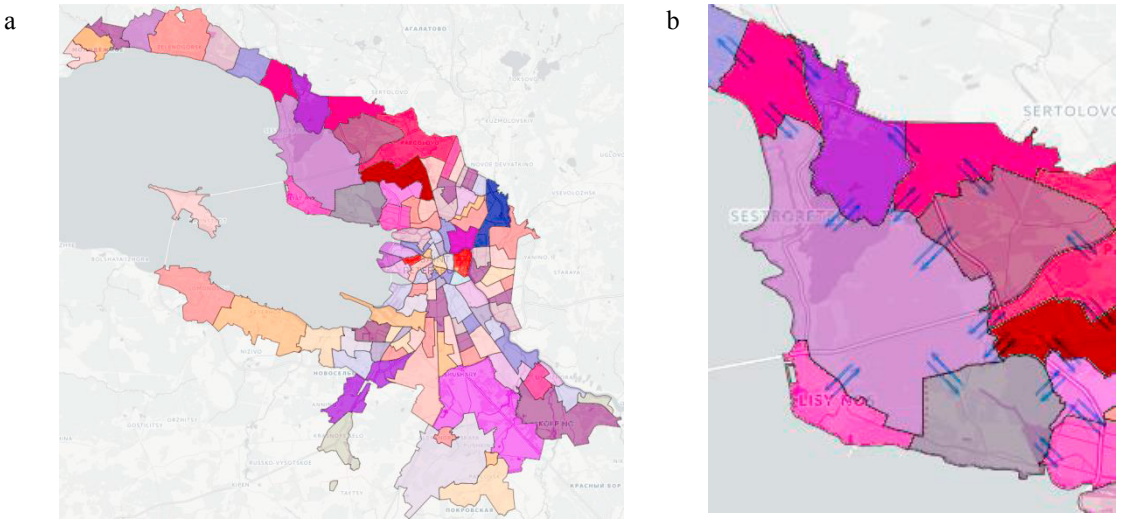

Fig. 2. Saint Petersburg city municipalities and connection forming logic: (a) Saint Petersburg city municipalities; (b) Connection forming schema.

In Fig. 3 below one can see results of comparison the algorithms.

It can be seen the proposed algorithm is able to deliver better redistribution of clusters on computational infrastructure. Initial improvements when an insignificant quantity of free cores is available is almost ~20%. It can be explained by the fact that enclosures have significant bunches of free cores and probability for highly connected districts to lay in the same enclosure is high enough. Small variations in performance during free cores increasing, in the end, is a result of heterogeneity in the structure of connections between clusters. The round-robin algorithm doesn't know anything about the connections between the clusters and their characteristics and may put highly connected clusters in separate enclosures, because of sequential splits it makes when it assigns a bunch of clusters to an enclosure.

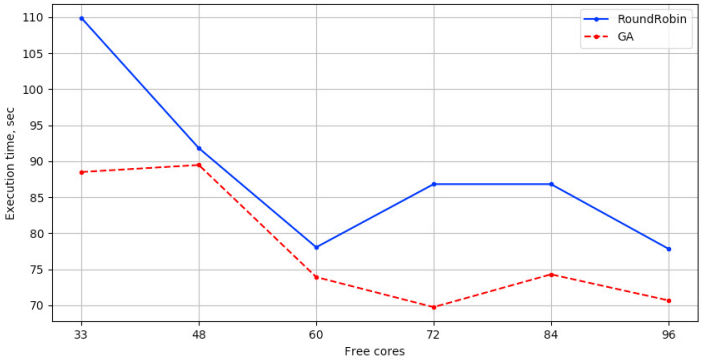

Fig. 3. The resulting execution time of simulation with load balancing performed by the proposed GA-based algorithm and the round-robin algorithm.

Decreasing of free cores quantity saves an improvement in performance in comparison to round robin. This effect is achieved by adjusting of clusters structure to structure of the computational environments when weakly connected clusters can be redistributed across different enclosures and highly connected should be brought as close as possible. It should be noted that efficiency of such matching strongly depends on characteristics of the environment and simulation itself.

If we are able to reduce the size of an agent to only 300 bytes the improvement gained by the genetic algorithm is dramatically reduced, and the round-robin algorithm performs good enough. That can be viewed in Fig 4.

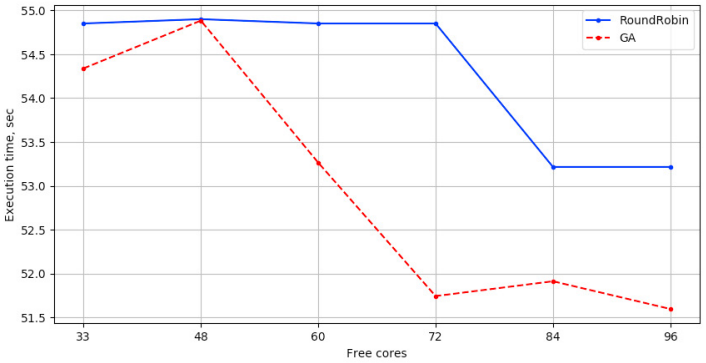

Fig. 4. The resulting execution time of simulation with reduced to 300 bytes agents size with load balancing performed by the proposed GA-based algorithm and the round-robin algorithm

5. Conclusion and future work

In work, the algorithm for optimization of matching between the model structure (in executable blocks of a distributed programming implementation of the model) and network structure of computational environment have been proposed. The algorithm shows significant improvements in comparison with the standard algorithm in a set of cases (up to ~20%) and can be used for organization of computationally efficient simulations in highly heterogeneous environments.

In the future work, the performance of the algorithm should be investigated under dynamic conditions when the clusters can reorganize during the simulation. Additional efforts should be paid to the research of multiscale models that can be combined into multilayered graphs and thus impose greater restrictions for neighbors assignment on computational nodes.

Acknowledgements

This research is financially supported by The Russian Science Foundation, Agreement №17-71-30029 with co-financing of Bank Saint Petersburg.

References

[1] C. E. Leiserson, “Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing,” *IEEE Trans. Comput.*, vol. C-34, no. 10, pp. 892–901, 1985.

[2] A. DeHon, “Compact, multilayer layout for butterfly fat-tree,” *SPAA 2000 Proc. twelfth Annu. ACM Symp. Parallel algorithms Archit.*, vol. 2, pp. 206–215, 2000.

[3] G. Faanes *et al.*, “Cray cascade: A scalable HPC system based on a dragonfly network,” *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC*, 2012.

[4] N. Butakov and D. Nasonov, “Co-evolutional genetic algorithm for workflow scheduling in heterogeneous distributed environment,” *2014 IEEE 8th Int. Conf. Appl. Inf. Commun. Technol.*, pp. 1–5, 2014.

[5] K.-I. Tsubota, S. Wada, H. Kamada, Y. Kitagawa, R. Lima, and T. Yamaguchi, “A Particle Method for Blood Flow Simulation, – Application to Flowing Red Blood Cells and Platelets –,” *J. Earth Simulator*, vol. 5, pp. 2–7, 2006.

[6] B. Cosenza, G. Cordasco, R. De Chiara, and V. Scarano, “Distributed Load Balancing for Parallel Agent-based Simulations.”

[7] Y. Wang, M. Lees, W. Cai, S. Zhou, and M. Y. H. Low, “Cluster based partitioning for agent-based crowd simulations,” in *Proceedings - Winter Simulation Conference*, 2009, pp. 1047–1058.

[8] V. Karbovskii, D. Voloshin, A. Karsakov, A. Bezgodov, and A. Zagarskikh, “Multiscale agent-based simulation in large city areas: emergency evacuation use case,” *Procedia - Procedia Comput. Sci.*, vol. 51, pp. 2367–2376, 2015.

[9] T. Mao, H. Jiang, J. Li, Y. Zhang, S. Xia, and Z. Wang, “Parallelizing continuum crowds,” *Proc. 17th ACM Symp. Virtual Real. Softw. Technol. - VRST ’10*, p. 231, 2010.

[10] M. J. Quinn, R. A. Metoyer, K. Hunter-zaworski, and C. Science, “Parallel implementation of the social forces model,” *Proc. Second*

Int. Conf. Pedestr. Evacuation Dyn., no. Figure 1, pp. 1–12, 2003.

[11] Y. Wang, M. Lees, and W. Cai, "Grid-based partitioning for large-scale distributed agent-based crowd simulation," in *Proceedings - Winter Simulation Conference*, 2012, pp. 1–12.

[12] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *2006 Work. Work. Support Large-Scale Sci. Work. '06*, vol. 14, pp. 217–230, 2006.

[13] D. Nasonov and N. Butakov, "Hybrid scheduling algorithm in early warning systems," *Procedia Comput. Sci.*, vol. 29, pp. 1677–1687, 2014.

[14] M. Mitchell, "An introduction to genetic algorithms," *Comput. Math. with Appl.*, vol. 32, no. 6, p. 133, 1996.

[15] K. Bochenina, N. Butakov, and A. Boukhanovsky, "Static scheduling of multiple workflows with soft deadlines in non-dedicated heterogeneous environments," *Futur. Gener. Comput. Syst.*, vol. 55, pp. 51–61, 2016.

[16] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3. pp. 375–385, 1996.