A Framework for Distributing Agent-Based Simulations

Gennaro Cordasco[2], Rosario De Chiara[1], Ada Mancuso[1], Dario Mazzeo[1], Vittorio Scarano[1], and Carmine Spagnuolo[1]

[1] ISISLab - Dipartimento di Informatica, Università degli Studi di Salerno
84084 Fisciano (SA) - Italy
{dechiara,vitsca}@di.unisa.it
[2] Dipartimento di Psicologia, Seconda Università degli Studi di Napoli
81100 Caserta - Italy
gennaro.cordasco@unina2.it

Abstract. Agent-based simulation models are an increasingly popular tool for research and management in many, different and diverse fields. In executing such simulations the "speed" is one of the most general and important issues. The traditional answer to this issue is to invest resources in deploying a dedicated installation of dedicated computers. In this paper we present a framework that is a parallel version of the MASON, a library for writing and running Agent-based simulations.

Keywords: Agent-based simulation, Heterogeneous Computing, Distributed Systems , Load-Balancing.

1 Introduction

The traditional answer to the need for HPC is to invest resources in deploying a dedicated installation of dedicated computers. Such solution can provide the computing power surge needed for highly specialized customers. Nonetheless a large amount of computing power is available, unused, in common installations like educational laboratories, accountant department, library PCs.

In this paper we present the architecture and report the performances of D-MASON, a parallel version of the MASON [9] library for writing and running simulations of Agent-based simulation models (ABMs). D-MASON is designed to harness the amount of unused computing power available in the scenarios above described. The intent of D-MASON is to provide an effective and efficient way of parallelizing MASON programs: effective because with D-MASON *you can do more* than what you can do with MASON; efficient because in order to obtain this additional computing power the developer has to do some incremental modifications to the MASON applications he has already written without re-designing them.

As in Condor [17] the purpose of D-MASON is to "harness wasted CPU power from otherwise idle desktop workstations" and to let the developer to look at such PCs as a platform composed by heterogeneous machines and the subdivision of the work among these machines takes into account such heterogeneity.

ABMs are an increasingly popular tool for research and management in many, different and diverse fields such as biology, ecology, economics, political science, sociology, etc.. The computer science community has responded to the need for tools and

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 460–470, 2012.

platforms that can help the development and testing of new models in each specific field by providing tools, libraries and frameworks that speed up and make easier the task of (massive) simulations.

Agent-based simulation toolkits are described in [10]. An interesting comparison is presented in [2]. Another interesting tool is Repast [13] with some studies on distributing the workload in [4]. It must be said that, in literature, MASON is recognized as one of the most useful and interesting simulation toolkits.

1.1 A Distributed Framework for Simulations

Among the motivations to our focus on distributing the simulation on a cluster of (homogeneous) machines, we can underline how the need for efficiency among the Agent-Based modeling tools is well recognized in literature: many reviews of state-of-the-art frameworks [2,12,15] place "speed" upfront as one of the most general and important issues. While a consistent work has been done to allow the distribution of agents on several computing nodes (see for recent examples [11,14]), our approach here is to introduce the distribution at the framework level in order to let the user to transparently harness the additional computing power.

This approach allows to hide to the user the most of the details of the implementation and in this way D-MASON requires just a moderate number of modifications into the source code achieving a good backward-compatibility with pre-existing MASON applications.

Because of the very experimental nature of complex social simulations there is always the need for a viable and reliable infrastructure for running (and keeping the records of) several experiments together the entire test settings. The resources needed to run and store results of such amount of experimental runs can be cheaply ensured only by a cluster, since the nature of interactive experiments, led by the social scientists with their multidisciplinary team, requires interaction with the computing infrastructure, which is often extremely expensive and technically demanding to get from supercomputing centers (that may, in principle, provide massive homogeneous environment).

In this scenario, our goal is to offer to such scientists a setting where a traditional MASON program can be run on one desktop, first, but can immediately harness the power of other desktops in the same laboratory by using D-MASON, thereby providing scaling up the size they can handle or significantly reduce the time needed for each iteration. The scientist, then, is able to run extensive tests by enrolling the different machines available, maybe, during off-peak hour.

Of course, it means that the resulting distributed system, collecting hardware from research labs, administration offices, etc. is highly heterogeneous in nature and, then, the challenge is how to use efficiently all the hardware without an impact on the "legitimate" user (i.e., the owner of the desktop) both on performances and on installation/customization of the machine. On the other hand, we would like that the program in MASON should not be very different than the corresponding program in D-MASON so that the scientist can easily modify it to run over an increasing number of hosts.

The rationale: The design of D-MASON is inspired by the need for efficiency, in a setting where computing resources are scarce, heterogeneous, not centrally managed and

that are used for other purposes during other periods of the workday. The compromises between efficiency and impact are reached with good results about performances, as witnessed by the tests we report, as well as the impact on the program in D-MASON is minimal, as described later in the paper.

2 MASON

MASON toolkit is a discrete-event simulation core and visualization library written in Java, designed to be used for a wide range of ABMs. The toolkit is written, using the standard Model-View-Controller (MVC) paradigm, in three layers: the *simulation* layer, the *visualization* layer and the *utility* layer. The simulation layer is the core of MASON and is mainly represented by an event scheduler and a variety of fields which hold agents into a given simulation space. MASON is mainly based on step-able agent: a computational entity which may be scheduled to perform some action (step), and which can interact (communicate) with other agents. The visualization layer permits both visualization and manipulation of the model. The simulation layer is independent from the visualization layer, which allows us to treat the model as a self-contained entity.

MASON was written with the aim of creating a flexible and efficient ABM which assures the complete reproducibility of results across heterogeneous hardware. This reproducibility feature is considered as a priority for long simulations (it allows to stop a simulation and move it among computers).

We decided to work on a distributed version of MASON for several reasons: MASON is one of the most expressive and efficient (as reported by many reviews [2,12,15]); MASON structure clearly separates visualization by simulation, making it particularly well suited to the re-engineering into a distributed "shape" of the framework. Another reason is the significant amount of research and simulations already present in the framework, which makes it particularly cost effective for the social scientist. The programmer is asked to use the new distributed version of some classes to transparently transform its already written simulation to a distributed simulation (e.g. extend `DistributedState` instead of `SimState`).

3 D-MASON: Distributed MASON

In the following we view ABMs as step-wise computations; i.e., agents behavior is computed in successive steps named *simulation step*. D-MASON is based on a master/workers paradigm (see Fig. 1): the master assigns a portion of the whole computation (i.e., a set of agents) to each worker. Then for each simulation step, each worker simulates the agents assigned and sends back the result of its computation to each interested worker.

Before presenting the architecture of D-MASON, in the next subsection will present the problems we faced in developing our distributed architecture.

3.1 Issues in D-MASON

Field partitioning. The problem of decomposing a program to a set of heterogeneous processors (workers) has been extensively studied (see [8] for a comprehensive presentation). In the case of ABMs a simple way to partition the whole work into different

tasks is to assign a fixed number of agents (proportional to the power of the worker) to each available worker. This approach named agents partitioning allows a balanced workload but introduce a significant communication overhead (since, at each step, agents can interact-with/manipulate other agents, an all–to–all communication is required). By noticing that most ABMs are inspired by natural models, where agents limited visibility allow to bound the range of interaction to a fixed range named agent's Area of Interest (AOI), several space partitioning approaches have been proposed [5,18,19] in order to reduce the communication overhead. In D-MASON, the space to be simulated (D-MASON field) is partitioned into regions. Each region, together with the agents contained are assigned to a worker. Since the AOI radius of an agent is small compared with the size of a region, the communication is limited to local messages (messages between workers, managing neighboring spaces, etc.).

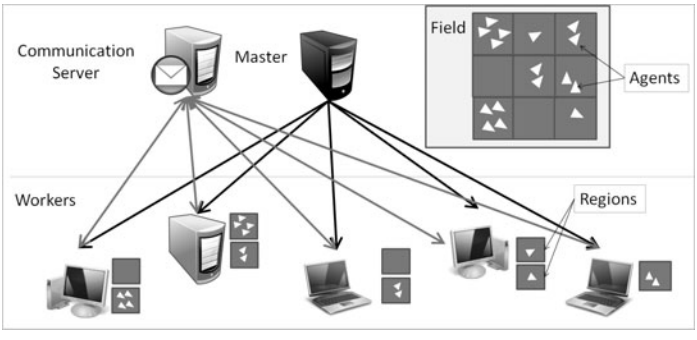

Fig. 1. D-MASON functional blocks

Synchronization. In order to guarantee the consistency of parallel implementation with respect to the sequential one, each worker needs to collect information about the neighboring regions. Each simulation step is formed by two phases: *communication/ synchronization* and *simulation*. First of all the worker sends to its neighbors (i.e., the workers responsible for its neighbor regions) the information about the agents that: are migrating to them; or may fall into the AOI of their agents. This information exchange is locally synchronized in order to let the simulation run consistently. We use a standard approach to achieve a consistent local synchronization of the distributed simulations. Each step is associated with a fixed state of the simulation. Regions are simulated step by step. Since the step i of region r is computed by using the states $i-1$ of r's neighborhood, the step i of a region cannot be executed until the states $i-1$ of its neighborhood have been computed and delivered. In other words, each region is synchronized with its neighborhood before each simulation phase.

Communication. D-MASON uses a well-known mechanism, based on the publish–subscribe design pattern, to propagate agents state information: a multicast channel is assigned to each region; users then simply subscribe to the channels associated with the

regions which overlap with their AOI to receive relevant message updates. The current version of D-MASON uses Java Message Service (JMS) for communication between workers. We use a special machine that run an Apache ActiveMQ Server [1] and acts as a JMS provider (i.e., it allows to generate and manage multicast channels and route messages accordingly). D-MASON however, is designed to be used with any Message Oriented Middleware that implements the publish–subscribe pattern. By providing a mapping between the abstract D-MASON's mechanism and the concrete implementation, it is possible, for instance, to use Scribe [3], a fully decentralized application-layer multicast built on top of the DHT Pastry [6]. Of course, also other simpler communication protocols can be used (such as sockets, Remote Method Invocation, etc.) but the effort of the programmer will be more consistent, since a mapping between a semantically rich paradigm, such as the publish–subscribe, and a simpler communication mechanism (stream, remote invocations, etc.) is needed.

Reproducibility. In order to guarantee an easy parallelization and to assure the reproducibility of results, paramount objective of the research areas interested in the ABMs, it is important to design the simulation is such a way that agents evolves simultaneously. Said in other words, during each simulation step, each agent computes its state at step i based on the state of its neighbors at step $i-1$. Thereafter all the agents updates their state simultaneously. Using this approach the simulation become embarrassingly parallelizable (there are no dependencies between agents' state), each simulation step can be executed in parallel overall the agents. Moreover, using this approach the order in which agents are scheduled does not affect the reproducibility of results[1].

Heterogeneity. D-MASON uses a simple but efficient technique to cope with heterogeneity. The idea is to clone the software run by high capable workers so that they could serve as multiple workers; i.e., a worker that is x times more powerful than other workers could execute x virtual workers (that is, simulating, concurrently, x regions).

3.2 Architecture

D-MASON adds a new layer named *D-simulation* which extends the MASON simulation layer. The new layer adds some features to the simulation layer that allows the distribution of the simulation work on multiple, even heterogeneous, machines. Notice that the new layer does not alter in any way the existing layers. Moreover, it has been designed so as to enable the porting of existing applications on distributed platforms in a transparent and easy way.

D-MASON architecture is divided into three functional blocks: *Management*,*Workers* and *Communication* (see Fig. 1). The Management layer provides a master application which will be used for coordinating the workers, handle the bootstrap and running the simulation. The master is responsible for partitioning the field into regions and assigning them to workers. Currently in D-MASON there are two types of field partitioning:

[1] Some simulations, especially those that evolve using a randomized approach, still require a mechanism that allows to schedule agents always in the same order, to obtain the reproducibility of results.

horizontal, where the division is done by splitting the field along one axis and square, where the division is done by using a grid. When all the parameter are set it is possible to start and interact with the entire distributed simulation (e.g. play, pause, stop). The workers are in charge of: simulating the agents that belongs to the assigned regions; handling the migration of agents; managing the synchronization between neighboring regions. Workers communicate by using the communication layer which provides a publish–subscribe mechanism.

D-MASON is available at `http://www.isislab.it/projects/dmason/`.

4 Testing

We performed a number of tests of D-MASON in order to assess both its ability to run simulations that are impractical or impossible to execute on a single computer (e.g., for CPU or memory requirements), its scalability and its effectiveness on exploiting heterogeneous hardware.

Setting of the Experiments. Simulations were conducted on a scenario consisting of five different type of hosts/workers described in the following table.

	P4	Xeon	Opt	i5	i7
CPU	1 x P4 3.4GHz	2 x Xeon 2.67GHz	2 x Opteron 1.9 MHz	1 x dual core i5 2.53 MHz	1 x quad core i7 2.53 MHz
RAM	2GB	3GB	4GB	4GB	8GB

The DFlockers testbed. We have performed our tests on *DFlockers* (see Fig. 2 right) by considering more than 50 different test settings. *DFlockers* is the distributed version of *Flockers* (see Fig. 2 left) which implements the *boid model* [16]. In the *boid model* each agent, named *boids* (from *birdoid*), gets instilled a range of *behaviors*. The behaviors are, in the most of cases, simply geometric calculations that every boid makes, considering the nearest boids it is flying with: for example the behavior called *pursuit* just let the boid to pursuit a moving target (e.g. another boid). Boids react to their neighbors so they must be able to identify them by filtering nearby boids out of the whole population. The brute force approach to this filtering consists in a $O(n^2)$ proximity screening and for this reason the efficiency of the implementation is yet to be considered an issue.

The boid model is designed for the aggregate motion of a simulated flock of boids as the result of the interactions of the relatively simple behaviors but, for the purposes of the test, this simulation reproduces common problems that must be taken into account in every other simulation: the search for nearest neighbors and a phase during which each agents updates its state. In the following we will use *Flockers* to indicate the MASON version of the simulation and with *DFlockers* we will indicate its D-MASON version.

Each test setting is characterized by the choice of the following parameter: number of agents (the size of the field is updated accordinglyin order to maintain a fixed density); regions-workers configuration, which establishes: the granularity of the field decomposition (i.e., the number of regions), the number of workers and the association between regions and workers.

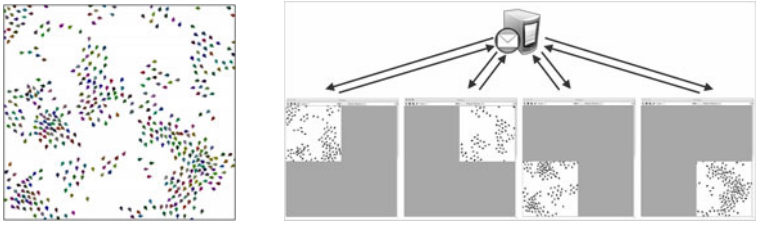

Fig. 2. (left) The *Flockers* testbed application. (right) *DFlockers* with 4 regions.

The configuration of each run of the test is represented by k couples (type of host, # of regions), where k is the number of hosts. For instance, the configuration $\{(\textbf{P4}, 1), (\textbf{Xeon}, 2), (\textbf{Opt}, 2), (\textbf{i7}, 4), (\textbf{i7}, 16)\}$ denotes a simulation run with $k = 5$ hosts (a **P4** simulating one region, a **Xeon** simulating two regions, an **Opt** simulating two regions and two **i7** simulating respectively 4 and 16 regions). Overall the field decomposition comprises $1 + 2 + 2 + 4 + 16 = 25$ regions.

In the following tests, each region is simulated by using a dedicated Java Virtual Machine (JVM). D-MASON also allows to simulate several regions on the same JVM by using different threads but the use of this approach requires tuning *each* virtual machine by increasing the amount of heap space accordingly to the number of regions simulated: this would be too burdensome to the heterogeneous setting we are envisioning, where we would like the users of PCs involved only marginally in the configurations. For sake of conciseness we opted for a one-JVM-one-region assignment since this represents also the worst-case for D-MASON (e.g. the overhead of the JVM is paid for each region, even on the same host). This decision also allowed us to uniquely indicate with JDK 1.6.0 update 25 the configuration of the JVM.

The communication is managed by a dedicated host running Apache ActiveMQ Server. Master, Workers and the Communication Server are connected using a standard 100Mbit LAN network (see Fig. 1). Each run is composed by 100 simulation steps and we used the average of each step running time, in our comparisons while the variance of such values was not significant.

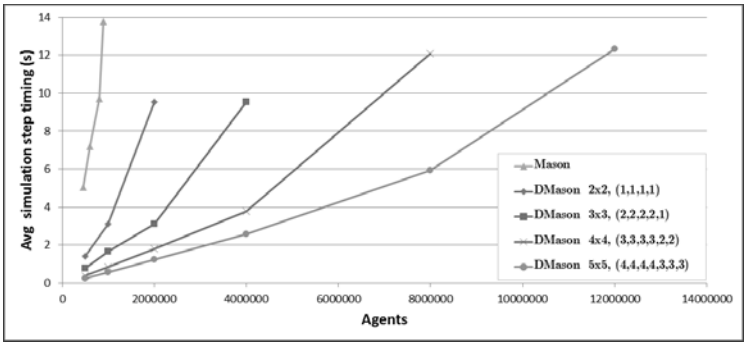

Fig. 3. D-MASON performances

4.1 Beyond MASON Limits

We first tested the limits of *Flockers* on MASON and we found that it is not possible to simulate more than $900,000$ agents. Moreover, using our best host (i.e., the **i7**), the simulation with $900,000$ agents took about 12 seconds for each simulation step. The purpose of this test is to show that the limits of MASON can easily be overcome by using a small number of workers. We performed a set of simulations with D-MASON on up to 7 homogeneous (**i7**) hosts (see Fig. 3). Results show that: (1) D-MASON is able to simulate far more agents than MASON does. The configuration $\{(\textbf{i7}, 4), (\textbf{i7}, 4), (\textbf{i7}, 4), (\textbf{i7}, 4), (\textbf{i7}, 3), (\textbf{i7}, 3), (\textbf{i7}, 3)\}$ allows to run a *DFlockers* simulation with $12,000,000$ agents and an average simulation step timing around 12 seconds; (2) D-MASON scales pretty well, using a fine grained field partitioning, as the number of workers increases we are able to increase D-MASON performances.

We have also analyzed the workload of the communication layer in order to check whether it may represent a bottleneck for the whole system performances. Clearly, we discovered that the communication cost of simulations increases proportionally to the number of regions used. However, our tests report that the workload on the communication server was always reasonably low and, therefore, it would be possible to further increase the granularity of partitioning and thus improve both the degree of parallelism, and load balancing.

4.2 Exploiting Heterogeneity

We performed another test with the aim of assessing D-MASON capability on exploiting heterogeneous platforms. Since the simulation is locally synchronized after each step, the application advances with the same speed provided by the slower worker/region in the system. For this reason it is necessary to configure the system in order to balance the load between the workers. In this test the field partitioning is always 5×5 square (25 regions), while we tested seven different configurations with five hosts and two values of the number of agents ($3,000,000$ and $5,000,000$). We decided to use very different hosts with the aim of showing that by adding a set of few old (usually unused) machines to a very powerful machine, one can measure sensibly improved performances (see Fig. 4). We will shortly discuss each of the bar in the figure: (a) D-MASON performances using only one **i7** machine; (b, c) by assigning regions to workers proportionally to each worker computational capability, it is possible to significantly improve performances; (d, e) gave the best performance with both $3,000,000$ (improvement 24%) and $5,000,000$ (improvement 28%) of agents; (f) reveals that one of the slowest machines has reached its limits and this badly reflects on overall simulation speed; (g) is the worst case in which the distribution of the region is uniform among machines, the slowest machines slow down the simulation while the fastest machines waste time waiting for synchronization.

The higher is the granularity of the partitioning, the better is the balancing that can be achieved. For instance, running the same simulation with a coarser granularity (e.g., 3×3 partition) would not allow to exploit the computational power of slower machines (each region is too computationally expensive for such machines).

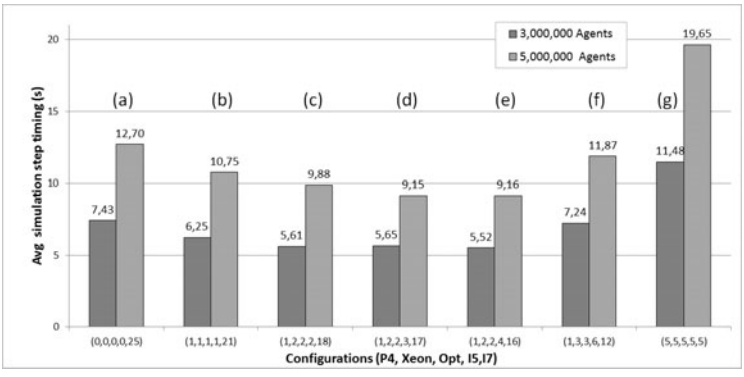

Fig. 4. D-MASON on a heterogeneous system. Seven different configurations are compared on *DFlockers* with $3,000,000$ and $5,000,000$ agents.

4.3 Fixed Timing Analysis

Iso-timing tests are justified by the need of having an interactive tool that allows faster development, debug and analysis of complex simulations. We intend to measure the maximum number of agents at a fixed simulation step pace of one step per second. The test is designed to measure this value on up to 7 homogeneous (**i7**) hosts while the field partitioning is always 5×5 square (25 regions). Regions are distributed uniformly among hosts. Then for each $h \in \{1, 2, 3, 4, 5, 6, 7\}$ we designed a set of simulations with h hosts. With each simulation we iteratively increased the number of agents until the average simulation step duration under 1 second. Under the conditions above in Fig. 5 we represent the performances of D-MASON.

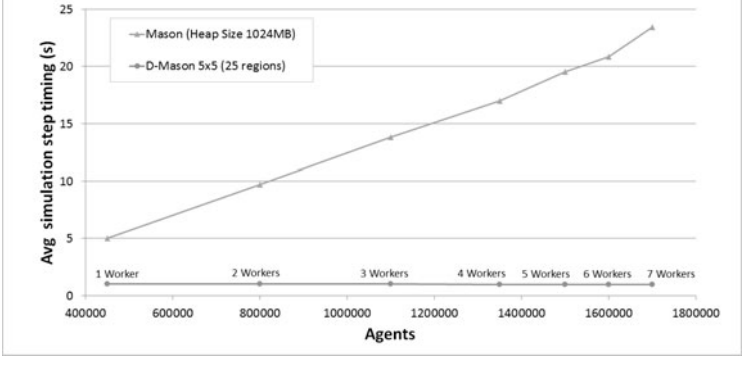

Fig. 5. Isotiming Analysis

In order to have a comparison for these results, we have performed a similar test on MASON. We used the same host (**i7**) and we increased the heap size of the JVM to 1024MB to let the machine to accommodate the increasing number of agents up to $1,700,000$. Obviously the performances of MASON fall down dramatically as along as

the number of agents increase. In the slowest configuration the average step completion time was around 25 seconds while, on 7 hosts D-MASON is 24 times faster. While this may reveal a superlinear speed up it is worth noting that each of the **i7** machines has a quad-core CPU. Of course D-MASON has been designed to exploit such configurations whose increasing presence seems to be a clear trend in the next years.

5 Conclusion and Future Work

D-MASON is a distributed version of MASON. MASON is a quite widespread framework for ABMs. Commonly to many kind of simulations, ABMs are CPU intensive applications and requires large amount of memory: these two characteristics put some limitations on the number and the complexity of the simulated agents. On the other hand the need for more complex simulations involving a large number of agents is always felt by researchers and practitioners. The rationale of developing D-MASON is to tackle this problem by providing a solution that does not require the user to rewrite his simulations and, nonetheless, pushing the limits of the maximum number of agents. This result is achieved by harvesting the unused CPU power usually largely available in installations like laboratories and by letting the computational work to be distributed among machines by addressing heterogeneity.

This paper reports on an currently undergoing project, and several issues are going to be tackled in the future. About the architectural level, first of all, we will further refine and try to optimize the thread-based version of D-MASON (i.e., when each region is a separate thread and not a separate JVM) by optimizing the thread management and providing short circuit of communication among regions on the same worker. This should further improve the already good results on performances. Then, we are planning to tackle load balancing issues by, first, allowing regions to migrate from particularly loaded workers to unloaded ones, and, then, modify the size and (possibly) the shape of regions on-the-fly.

About the simulation library, we are currently tackling other fields, since different categories (such as 3D, graph-based, etc.) do require suitably tailored approaches for the partitioning in regions. A long-term objective is also an improved management of parallelism, with distributed visualization and monitoring workers.

Finally, the project will be soon released under a Free and Open Software license.

References

1. Apache ActiveMQ, `http://activemq.apache.org/`
2. Berryman, M.: Review of Software Platforms for Agent Based Models. Technical Report DSTO-GD-0532, Australian Government, Department of Defence (2008)
3. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC) 20, 100–110 (2002)
4. Cicirelli, F., Furfaro, A., Giordano, A., Nigro, L.: Distributed Simulation of RePast Models over HLA/Actors. In: Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT 2009, pp. 184–191. IEEE Computer Society, Washington, DC (2009)

5. Cosenza, B., Cordasco, G., De Chiara, R., Scarano, V.: Distributed load balancing for parallel agent-based simulations. In: Proc. of the 19th Euromicro Inter. Conf. on Parallel, Distributed and Network-Based Computing, PDP 2011 (2011)
6. Druschel, P., Rowstron, A.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Proc. of the 18th IFIP/ACM Inter. Conference on Distributed Systems Platforms (Middleware 20), pp. 329–350 (November 2001)
7. Epstein, J.M.: Generative Social Science: Studies in Agent-Based Computational Modeling. Princeton University Press (2007)
8. Hwang, K., Xu, Z.: Scalable Parallel Computing: Technology, Architecture, Programming. McGraw-Hill (1998)
9. Luke, S., Balan, G.C., Panait, L., Cioffi-Revilla, C., Paus, S.: MASON: A Java Multi-Agent Simulation Library. In: Proceedings of the Agent 2003 Conference, Chicago, IL, October 2 - October 4 (2003)
10. Macal, C.M., North, M.J.: Tutorial on agent-based modeling and simulation part 2: how to model with agents. In: Proceedings of the 38th Conference on Winter Simulation, WSC 2006, pp. 73–83 (2006)
11. Mengistu, D., Troger, P., Lundberg, L., Davidsson, P.: Scalability in Distributed Multi-Agent Based Simulations: The JADE Case. In: Proc. Second Int. Conf. Future Generation Communication and Networking Symposia FGCNS 2008, vol. 5, pp. 93–99 (2008)
12. Najlis, R., Janssen, M.A., Parkerx, D.C.: Software tools and communication issues. In: Parker, D.C., Berger, T., Manson, S.M. (eds.) Proc. Agent-Based Models of Land-Use and Land-Cover Change Workshop, pp. 17–30 (2001)
13. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the Repast agent modeling toolkit. ACM Trans. Model. Comput. Simul. 16, 1–25 (2006)
14. Pawlaszczyk, D., Strassburger, S.: Scalability in distributed simulations of agent-based models. In: Proc. Winter Simulation Conf. (WSC) the 2009, pp. 1189–1200 (2009)
15. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: Review and development recommendations. Simulation 82, 609–623 (2006)
16. Reynolds, C.: Steering behaviors for autonomous characters (1999)
17. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurr. Comput.: Pract. Exper. 17, 323–356 (2005)
18. Zhang, Y., Mueller, F., Cui, X., Potok, T.: Large-Scale Multi-Dimensional Document Clustering on GPU Clusters. In: IEEE International Parallel and Distributed Processing Symposium (2010)
19. Zhou, B., Zhou, S.: Parallel simulation of group behaviors. In: WSC 2004: Proceedings of the 36th Conference on Winter Simulation, pp. 364–370 (2004)