Journal of Network and Computer Applications 122 (2018) 115–127

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Parallelization of space-aware applications: Modeling and performance analysis

Franco Cicirelli, Agostino Forestiero, Andrea Giordano, Carlo Mastroianni *

ICAR-CNR, via P. Bucci 7/11C, Rende (CS), Italy

ARTICLE INFO

Keywords:
Space-aware applications
Petri Nets
Parallel applications
Performance evaluation
Multi-agent systems

ABSTRACT

Many applications in fields like sociology, biology and urban computing, need to cope with an explicit use of a spatial environment, or territory. Such applications, referred to as space-aware applications (SAAs), are based on a set of entities that live and operate in a territory. Parallel execution of space-aware applications is needed to improve the performance when the demand of computational resources increases. Despite the great interest towards SAAs, there is a lack of models and theoretical results for assessing and predicting their execution performance. This paper presents a novel framework, based on Stochastic Time Petri nets, which is able to capture the execution dynamics of parallel SAAs, and model the aspects related to computation, synchronization and communication. The framework has been validated by comparing the predicted performance results for a testbed application, i.e., the ant clustering and sorting algorithm, to those experienced on a real execution platform. An extensive set of experiments have been performed to analyze the impact on the performance of some important parameters, among which the number of parallel nodes and the ratio between computation and communication load.

1. Introduction

Space-aware applications (SAAs) are applications that rely on the explicit representation of a territory, that is, a spatial environment on which data and objects are defined (Amouroux et al., 2007; Shook et al., 2013). To improve the execution efficiency and the scalability of complex space-aware applications, the computation can be parallelized on distributed nodes, and the partitioning can be driven by the topological properties of the territory itself. Specifically, it is possible to assign different regions of the territory to different computing nodes which can process local data in parallel. Urban-computing, geology, biology, hydrology, social sciences, logistics and transportation, smart electrical grids, are significant examples of application fields strongly related to SAAs (Cicirelli et al., 2016a; Gong et al., 2013; Tang et al., 2011; Garofalo et al., 2017).

SAAs are becoming increasingly recurrent also due to the rapid emergence of two relatively novel computation paradigms, specifically the “Internet of Things” (IoT) (Atzori et al., 2010; Lee and Lee) and some new distributed forms of computing, like Fog Computing and Edge Computing (Bonomi et al., 2012; Krishnan et al., 2015; Hu et al., 2017), where the computation is brought closer to final users and/or where the data to elaborate is produced. In these scenarios, it is natural to manage data through the use of computing entities distributed over the territory, in order to perform computation as close as possible to data sources and increase the performance. The multi-agent paradigm is commonly used in the literature for the modeling and implementation of SAAs (Amouroux et al., 2007; Shook et al., 2013): the computation is executed by agents (Wooldridge, 2002) that live and operate in the territory.

Usually, SAAs cannot be parallelized through an “embarrassingly parallel” approach (Ekanayake and Fox, 2010), i.e., computation at the single nodes cannot be performed in isolation because parallel tasks need to exchange data during computation. For example, a smart city application that analyzes the user mobility typically requires that the events occurred in a city neighborhood are communicated to the adjacent neighborhoods (Harri et al., 2009). In this context, a common challenge is to cope with synchronization and communication issues affecting the execution performance (Shook et al., 2013; Gong et al., 2013) of parallel SAAs, and ensure a continuous view of the territory despite its partitioning (Cordasco et al., 2013). More in general, there is a strong need for methodological approaches that help analyzing the performance of the parallel execution of space-aware applications.

* Corresponding author.
E-mail address: carlo.mastroianni@icar.cnr.it (C. Mastroianni).

https://doi.org/10.1016/j.jnca.2018.08.015
Received 14 December 2017; Received in revised form 3 July 2018; Accepted 25 August 2018
Available online 30 August 2018

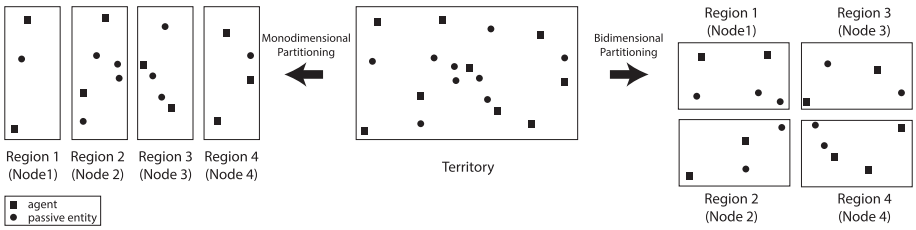


Fig. 1. A territory partitioned into regions which are associated with parallel computing nodes. Two alternative types of partitioning are shown, monodimensional and bidimensional.

In this paper, we present and discuss a framework that helps to assess and predict the performance of general parallel SAAs by taking into account the overheads of computation, synchronization and communication on the application performance. The framework permits to consider a territory partitioned into regions exploiting both monodimensional and bidimensional schemas. Each region, along with the local entities, data and computation, is assigned to a distinct computing node. The computation advances through successive *steps*, and synchronization and data exchange are required between two consecutive steps. The semantics of the distributed computation is captured by using a Stochastic Time Petri net (Murata, 1989; Paolieri et al., 2016) model, which represents the interactions among nodes, and specifically the synchronization aspects.

With our framework, it is possible to analyze and predict the performance of SAA applications in a large variety of scenarios. This objective is achieved as follows. Firstly, a set of tests are executed on a real platform, in order to estimate the random distributions of unit computation and communication times at single nodes. Afterwards, the obtained random distributions are used in order to predict, by simulation, the overall performance for different configurations of the system. We used two different simulators: a Petri net simulator, i.e., Yasper, which directly executes the Petri net model, and a purposely written Matlab simulator. The latter simulator reproduces the same behavior, but is faster and more flexible, as it permits to execute a parameter sweep analysis and to use any kind of distribution for computation and communication times, including those observed experimentally.[1]

In order to highlight the effectiveness of the proposed approach, the framework was exploited to analyze the performance, in terms of execution time, of the parallel execution of a well-known algorithm, the ant-based clustering and sorting algorithm (Deneubourg et al., 1990; Bonabeau et al., 1999). This algorithm was chosen because it is the ancestor of a large variety of bio-inspired algorithms based on the distribution of the computation among a large number of agents dispersed in a territory.

Beyond analyzing the ant-based clustering and sorting algorithm, we explain how the framework can be used to generalize the study and establish a general relationship between the application performance and a set of parameters, such as the number of nodes on which the computation is partitioned, the density of agents, the average computation and communication load of a single agent, the relative weights of computation and communication times. In addition, we compare the performance obtained with monodimensional and bidimensional territory partitioning, in order to help the application experts to individuate the most efficient option when both solutions are admissible. This kind of analysis can help to predict the performance when the parameter values are fixed, and to tune the behavior of the application when it is possible to act on the parameter values.

The rest of the paper is organized as follows: Section 2 illustrates how the parallelization is driven by space partitioning, and describes the involved issues, specifically those related to the data exchange among parallel nodes; Section 3 presents the model of parallel execution, formally represented by a Petri net, and analyzes the overhead induced by synchronization and communication; Section 4, after summarizing the ant clustering and sorting algorithm used for the evaluation, reports and discusses the experimental analysis and performance results obtained both in a real platform and through the presented framework; Section 5 discusses some relevant state-of-the-art on the parallel execution of space-aware applications. Finally, Section 6 concludes the paper and provides some indications on future research avenues.

2. Parallelizing execution through space partitioning

The aim of this section is to show how the territory partitioning drives the parallel execution of SAAs. Many SAAs applications can be profitably implemented on a bidimensional grid of *cells* (the territory) where each cell can contain *active* and *passive* entities. An active entity, also referred to as *agent* in the following, is able to perform autonomous computation, explore and modify the territory and move around it. A passive entity represent an object or data on which active entities perform the computation. The computation is assumed to advance in steps: at each *step*, all the agents perform their piece of computation. Each agent can operate and move only inside its *visibility area* individuated by defining the *visibility radius* as the maximum distance, in terms of cells, at which the agent can operate.

A natural way of optimizing the execution of algorithms working on spatial data is to partition the territory into *regions* and assign each region, along with the contained entities, to a different *computing node* that is in charge of performing the computation pertaining to that portion of the territory. Partitioning favors system scalability in that as the size of the territory increases, more computing nodes can be used to speed up the execution. A territory can be partitioned through either a *monodimensional* or a *bidimensional* schema, as shown in Fig. 1.

Since the visibility area of agents can span across multiple regions (see Fig. 2), an agent can require to access and/or manipulate data assigned to different computing nodes. To this purpose, a duplication mechanism can be adopted, which consists in replicating the edge areas of adjacent regions at the end of each step (Cosenza et al., 2011; Cicirelli et al., 2016a, 2016b). Such areas, referred to as *borders*, are kept aligned by exchanging, at each step, *update messages* among the adjacent computing nodes.

More in detail, the management of borders is performed as follows. The border area of a region is composed of two distinct parts: the *local border* and the *mirror border*. These two parts are portrayed in Fig. 3(a) and (b), which show the borders in the cases of monodimensional and bidimensional space partitioning, respectively. The local border is managed by the local node and its content, i.e., the active and passive entities, is replicated in the mirror border of the adjacent nodes. The agents located in a border area are mirrored by means of *phantom* agents (i.e., agents that hold the same information of original agents but do not perform computation), while passive entities are simply duplicated.

The parallel execution approach described so far is implemented at each node by performing the *execution loop* depicted in Fig. 4, where

[1] Both the Petri net model and the Matlab simulator are available at the URL http://apswarm.icar.cnr.it/applications/j2ee-modules/sc/jnca.

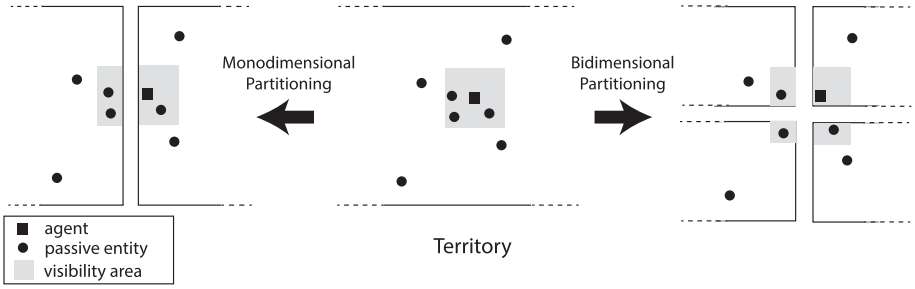
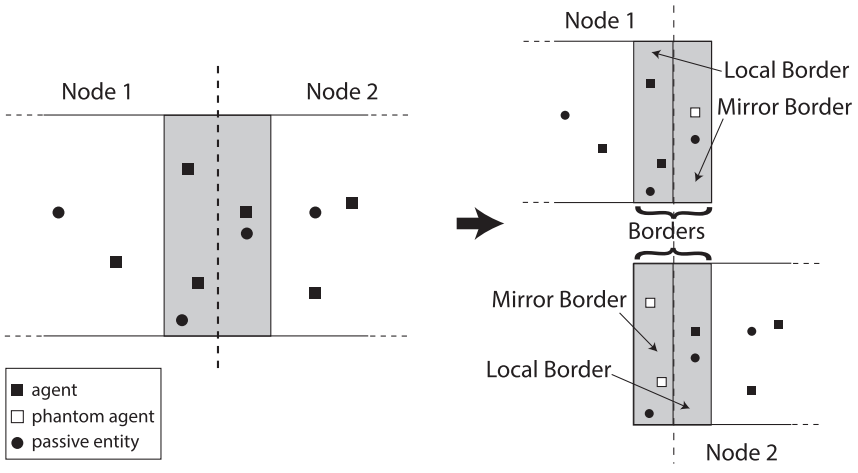
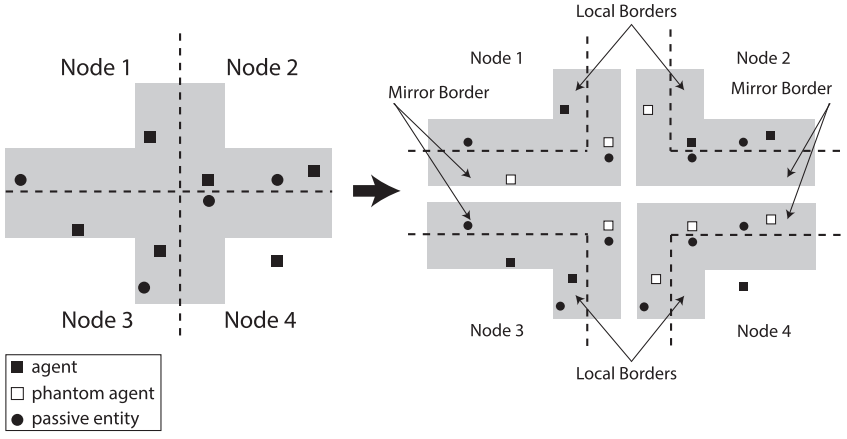


Fig. 2. Example of a territory partitioning in which the visibility area is split among two adjacent regions (monodimensional partitioning) and four adjacent regions (bidimensional partitioning).



(a) monodimensional partitioning



(b) bidimensional partitioning

Fig. 3. Border areas of adjacent nodes.

one iteration of the loop corresponds to the execution of an application step. The loop includes three phases: first, the node executes the local computation, which means that all the agents of the region perform their computation for the current step. Then, the node sends to its neighbor nodes the update messages and, finally, the node waits for the update messages coming from its neighbors.

Through the transmission of the update messages, a node communicates to its neighbors that it has completed the execution of the current step. Therefore, the coordination of parallel execution is performed through the exchange of update messages, which are used to implement a synchronization barrier between a node and its adjacent nodes: once the messages are received, the barrier is passed and the node can begin the execution pertaining to the next step.

The approach described so far, which will be formally modeled in the next section, refers to the cases where all the operations are local. It is possible to extend the approach so as to include “global” operations,

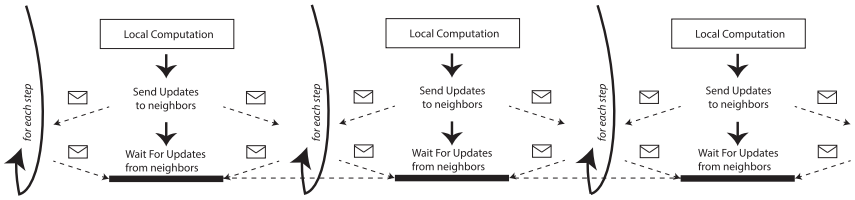


Fig. 4. Execution loop for a single application-step.

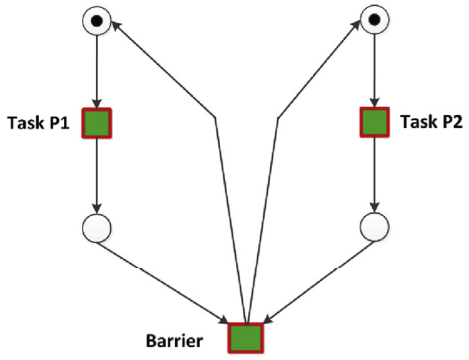


Fig. 5. A simple Petri net modeling two periodic processes which require a synchronization barrier.

i.e., operations that concern the whole territory and/or all the agents. For example, broadcasting an information to all the agents or updating some information on all the cells with a diffusive phenomenon. Global operations can be supported by combining the execution loop of single nodes (Fig. 4) with a synchronization barrier that involves all the nodes. This aspect has been addressed in (Cicirelli et al., 2016a) and is not the focus of this work.

3. A model for performance analysis

The efficiency of the computation carried out by exploiting the *step-based execution cycle* described in Section 2, can be assessed by measuring the average time needed to perform a step on all the nodes. Such time, in the following referred to as *average step time* and denoted as T_{step}, is obtained by dividing the execution time of the parallel application by the number of executed steps.

From Fig. 4 it is noticed that the average step time depends on the amounts of time spent, for a single step, on three different activities: (i) the *local computation time*, or simply *computation time*, i.e., the time spent to perform agent operations at single nodes; (ii) the *communication time*, i.e., the time needed to transmit update messages among adjacent nodes; (iii) the *synchronization time*, i.e., the time spent on synchronization barriers. The relationship among these quantities is formalized through a Petri net model illustrated in Section 3.2, after a brief introduction to Petri net which is given in Section 3.1. Successively, in Section 3.3, we discuss how the computation and communication times are related to the areas of the involved portions of the territory, i.e., of regions and region borders, respectively.

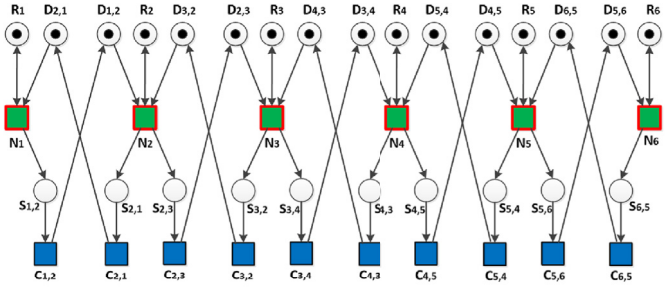
3.1. Petri Nets and performance analysis

Petri nets (Murata, 1989; Peterson, 1977) are a well-know formalism for the modeling and analysis of concurrent and distributed systems and are particularly suitable to capture synchronization, mutual exclusion and non-deterministic aspects. A Petri net is a bipartite and direct graph consisting of two kinds of nodes called *places* and *transitions*. A place can contain an arbitrary number of *tokens*. *Arcs* are used to connect places to transitions and transitions to places. Arcs are weighed, and in the following the weight of each arc is supposed to be one. Graphically, places are represented by empty circles, transitions by boxes, and tokens by filled small circles. A transition is *enabled* and can *fire* when all its incoming places hold at least one token. When a transition T fires, one token is *consumed* at each incoming (input) place of T, and one token is *produced* on each of the outgoing (output) places of T. As an example, Fig. 5 reports a simple Petri net that models two periodic processes $P1$ and $P2$ which, after executing their own tasks (modeled by using transitions *Task* $P1$ and *Task* $P2$, respectively), need to synchronize with each other before executing their tasks again. In this example, the synchronization barrier is modeled through the transition *Barrier*.

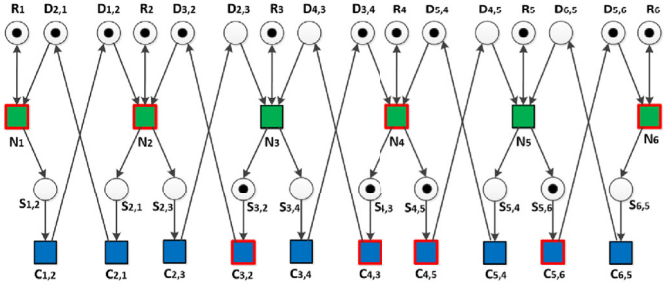
Some enhanced versions of Petri nets permit to add timing information to transitions so as to express their firing (or delay) time. Stochastic Time Petri Nets (Paolieri et al., 2016) allow to express the firing time of a transition as a random variable modeled by an arbitrary probability distribution function. This feature is exploited in this paper to analyze the performance of space-aware applications. The semantics of Stochastic Time Petri Nets is provided in Appendix A.

3.2. A petri net model for parallel execution of SAAs

The loop of the parallel execution introduced in Section 2 (see Fig. 4), which includes the synchronization barriers and the communication among neighbor nodes, is here formally modeled using the Petri net language. Fig. 6 reports the Petri net model for the monodi-



(a) All the nodes are ready to execute.



(b) After execution at N_3, N_4 and N_5, and data transmission from N_3 to N_4 and from N_5 to N_4.

Fig. 6. Petri net representing the local computation and data transmission among six parallel nodes, in the case of monodimensional partitioning.

mensional scenario in the case that the territory is partitioned into six regions, and the execution is parallelized on the six corresponding computing nodes $i \in \{1 \dots 6\}$. Six *transitions*, labeled as N_i, and depicted in green in the figure, are respectively associated with each computing node. The *firing* of a transition corresponds to the execution of the local computation *step* at the corresponding node.

Each transition N_i is connected by inbound arcs to three input *places*, and in accordance to Petri net firing semantics, the transition is *enabled*, and the local computation can start, if all the input places hold at least one *token*. The three input places are labeled as $D_{i-1,i}$, $D_{i+1,i}$ and R_i. The presence of a token in place $D_{i-1,i}(D_{i+1,i})$ means that the left(right) neighbor node has already delivered its update message to the node i. R_i is both an input and an output place of the transition N_i and it models the fact that the computation at a given step cannot start before the end of the computation at the previous step. Each transition N_i is also connected by outbound arcs to three output *places*. One of them is the place R_i described before. The other two output places are labeled as $S_{i,i-1}$ and $S_{i,i+1}$. A token in the place $S_{i,i-1}(S_{i,i+1})$ indicates that the computation at node i, for the current step, is completed and an update message has to be sent to the left(right) neighbor node. Indeed, $S_{i,i-1}$ and $S_{i,i+1}$ are the input places of two "communication" transitions $C_{i,i-1}$ and $C_{i,i+1}$, depicted in blue in Fig. 6, which model the transmission of the update messages from node i to the two neighbor nodes $i-1$ and $i+1$, respectively. When a communication transition $C_{i,i-1}(C_{i,i+1})$ fires, it produces a token in the output place $D_{i,i-1}(D_{i,i+1})$, which models that the neighbor node $i-1(i+1)$ has received the update message from node i. The transitions and places related to the two extreme regions (on the left and on the right) of the territory are modeled differently in order to consider that these regions have only one neighbor.

In summary, the Petri net models a set of cyclic synchronization barriers, whose semantics is that each node can execute a step when it has completed the execution at the previous step and it has received the data from the two neighbor nodes.

The initial state of the Petri net is depicted in Fig. 6(a), where all the input places of transitions N_i contain one token, meaning that these transitions are enabled, i.e., the computing nodes are ready to begin the execution of the first step. This initial marking ensures that the first operation at each node is the execution of computation, matching the execution loop shown in Fig. 4. In Fig. 6, the ability to fire a transition is represented by the presence of a red border on the box representing the transition. Fig. 6(b) represents the situation after the computation at nodes N_3, N_4 and N_5, and after the transmission of data from N_3 to N_4 and from N_5 to N_4. N_4 is now enabled to execute the next step, because it has performed the previous computation and has received the update messages from its neighboring nodes N_3 and N_5. In the Petri net model, this corresponds to the presence of three new tokens at the input places of N_4, which means that the synchronization barrier which precedes the next computation at node N_4 has been successfully passed. It is also noticed that nodes N_3 and N_5 are not yet enabled because they are still waiting for the data regarding the computation of the previous step at the neighbor nodes.

From the Petri net depicted in Fig. 6, with N computing nodes, it emerges that the time experienced at a generic node $i \in \{1, 2, \dots, N\}$ at the end of the step k, denoted ad $T_i(k)$, is determined by the recursive expression (1):

$$T_i(k+1) = \max(T_i(k),\ T_{i-1}(k) + C_{i-1,i}(k),\ T_{i+1}(k) + C_{i+1,i}(k)) + T_{comp_i}(k+1) \quad (1)$$

where $T_{comp_i}(k)$ is the time needed by node i to compute the step k, and $C_{m,n}(k)$ is the time needed to send an update message from node m to node n after the execution of step k. It is assumed that all the nodes begin the execution at the same time, i.e., $T_i(0) = 0$ for each node.[2]

[2] To let expression (1) be consistent for the nodes at the two extremes, the values of $T_0(k)$ and $T_{N+1}(k)$ are set to 0 for each step k.

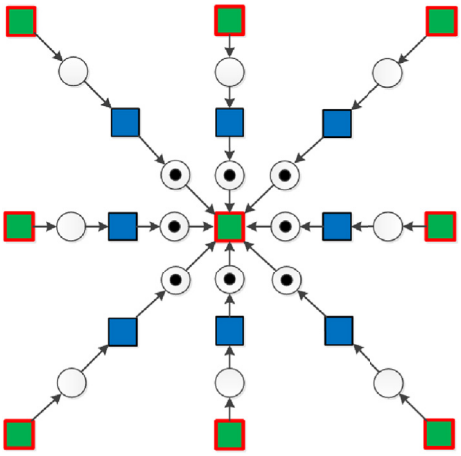


Fig. 7. Portion of a Petri net representing the computation and data transmission in the case of bidimensional partitioning.

We used this expression to simulate the Petri net with Matlab, as will be discussed in Section 4.3.

The Petri net model can be easily extended to formalize also the parallel execution in the case of bidimensional partitioning. In particular, the input and output places of each computing transition N_i need to be changed to take into account that in a bidimensional scenario the number of neighbor nodes is eight. The Petri net model for the bidimensional case cannot be easily depicted graphically. Fig. 7 contains an excerpt of the model, which shows that the execution at a node (the central node in the figure) requires the reception of information by the eight adjacent nodes.

The firing times of computation transitions, i.e., those depicted in green in Figs. 6 and 7, and of communication transitions, depicted in blue, are modeled through random variables, according to the definition of Stochastic Time Petri Nets (see Appendix A). The distributions of these random variables depend on the computation load assigned to each node and on the communication burden among nodes, as explained in the following.

3.3. Computation and communication load

The time needed to perform the local computations on the different regions, and to send the update messages among adjacent regions, are related to the portions of territory involved in computation and communication. For the sake of simplicity, in this paper we consider the case in which the density of agents is approximately uniform in the territory, even though the framework is applicable to scenarios where this assumption does not hold. In the considered case, the local computation time needed to execute the agent operations at a node is approximately proportional to the area of the local region. Analogously, the size of the update messages sent to neighbor nodes, and therefore the communication time, is approximately proportional to the area of the involved border: indeed, each node builds a message to collect the updates occurred in the local border area during the last time step.

For this reason, it is useful to compute how the areas of the regions and of the borders are related to the number of computing nodes N at which the parallel computation is performed, and to the value of the visibility radius, as defined in Section 2, referred to as r in the following. For the sake of simplicity, let us consider the scenario in which the territory has a rectangular shape, with length L and height H. We also assume that the territory is equally partitioned among the N nodes. The area of a single region is independent from the type of partitioning, monodimensional or bidimensional, and it is equal to A/N, where $A = L \cdot H$ is the area of the entire territory. Conversely, the area of the borders must be computed separately for monodimensional and

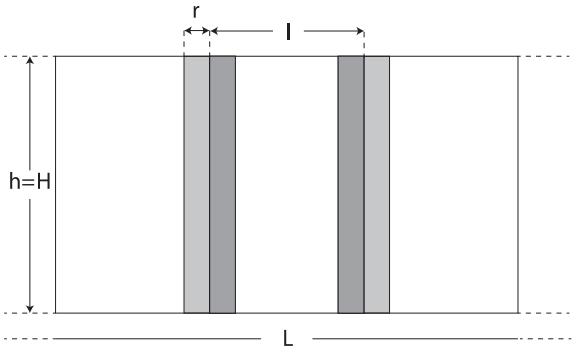


Fig. 8. Monodimensional space partitioning.

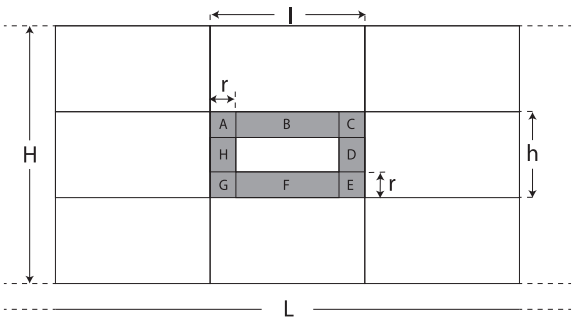


Fig. 9. Bidimensional space partitioning.

bidimensional partitioning.

In the case of monodimensional partitioning, shown in Fig. 8, the area of a single border is $H \cdot r$. It is noticed that this area depends only on the visibility radius and it is independent from the number of computing regions.

In the case of bidimensional partitioning, there are three types of border areas, depending on the relative positions of the two regions involved in the communication: (i) the area involved in north–south communication; (ii) the area involved in east–west communication and (iii) the area involved in diagonal communication. These areas are referred to as A_{NS}, A_{EW} and A_{Diag}, respectively. In Fig. 9, A_{NS} is the area of rectangles[3] *ABC* and *GFE*, A_{EW} is the area of rectangles *AHG* and *CDE*, and A_{Diag} is the area of the rectangles *A*, *C*, *E* and *G*. There can exist different possibilities to partition the territory and obtain the same number of nodes. Let P_h and P_v, the number of horizontal and vertical partitions of the territory, so that $N = P_h \cdot P_v$. The choice of the values of P_h and P_v determines the shape of the single regions and the size of the border areas. With reference to Fig. 9, the areas of the borders can be expressed as:

$$A_{NS} = (H/P_h) \cdot r \quad A_{EW} = (L/P_v) \cdot r \quad A_{Diag} = r \cdot r \tag{2}$$

It follows that the choice of P_h and P_v has an effect on the overall amount of border information that needs to be transmitted from a node to its adjacent nodes. This amount is proportional to the sum of the border areas A_B, which is $A_B = 2{*}A_{NS} + 2{*}A_{EW} + 4{*}A_{Diag}$. In can be easily derived that, given a number of nodes, the horizontal and vertical partitioning that minimizes the quantity A_B, is obtained when $P_h/P_v = H/L$ holds, i.e., when regions are square-shaped.[4] In the case of square-shaped territory the condition reduces to $P_v = P_h$.

This result, and in general the expressions for areas of regions and borders discussed in this section, will be used in the following to estimate the values of computation and communication times.

[3] Here, the notation "rectangle *ABC*" individuates the rectangle obtained as the union of rectangles *A*, *B* and *C*.

[4] Let $l = L/P_v$ and $h = H/P_h$ be respectively the length and the height of a region, as shown in Fig. 9. We express A_B in terms of l, by using the substitution $h = \frac{A}{l}$, where $A = l \cdot h$ is the area of the region. We obtain:

$$A_B(l) = 2(l + \frac{A}{l})r + 4r^2 \tag{3}$$

We evaluate the first derivative of $A_B(l)$ with respect to l and obtain:

$$\frac{dA_B(l)}{dl} = \frac{l^2 - A}{l^2} \cdot 2r \tag{4}$$

In order to find the minimum of $A_B(l)$, we set its first derivative equal to zero, which gives:

$$l = \sqrt{A} \tag{5}$$

Since the second derivative at $l = \sqrt{A}$ is positive, the minimum communication burden is achieved when the region has a square shape. This is obtained when $P_h/P_v = H/L$ holds. Of course, the admissible options for the space partitioning also depend on the number of nodes *N*, since there is the constraint $N = P_v \cdot P_h$.

4. Experimental analysis and performance results

In this section we present the performance results, obtained on a real platform, and discuss how these results can be generalized by adopting the model of parallel execution described in the previous section. The section is organized as follows: (i) in Section 4.1, we describe the classical ant-based clustering and sorting algorithm, which is used as a testbed; (ii) in Section 4.2, we report the performance results obtained on a real platform; (iii) in Section 4.3, we validate the Petri net model by showing that it is capable of reproducing and predicting the results observed in the real platform; (iv) finally, in Section 4.4 we use the model to generalize the analysis and predict the results in scenarios with different numbers of nodes and different relationships between computation and communication load.

4.1. The ant-based clustering and sorting algorithm

As an example of a space-aware application that can be parallelized through a territory partitioning approach, in this paper we use the ant-based clustering and sorting algorithm, inspired by the behavior of some species of ants that cluster corpses to form a "cemetery" or sort their larvae into separate piles. This algorithm can be considered as the ancestor of a large number of bio-inspired and swarm intelligence algorithms. The basic features of the clustering and sorting behavior of ants can be reproduced by a simple model in which agents move over a territory and pick up and deposit items on the basis of local information (Deneubourg et al., 1990; Bonabeau et al., 1999).

If items are all of the same kind, the goal of ant-based clustering is to create regions in which items are accumulated, leaving empty regions in between. If items belong to a number of different types, or classes, the objective becomes to sort items spatially, i.e., separate items of different classes and cluster items of the same class. In the following we refer to the sorting model, i.e., to the case in which items belong to different classes, since the clustering model can be considered as a special case of the sorting model.

The territory is modeled as a bidimensional grid of *cells*. Each agent has visibility over the items located in its own cell and in the cells distant no more than r cells, where r is the *visibility radius*. The set of cells defined in this way is referred to as *visibility area* of the agent. Each agent contributes to the spatial sorting of items by *picking* and *dropping* items from/to the cells. At each step, every agent moves randomly in the environment, towards an adjacent cell, and in the new cell performs a drop or pick attempt, according to whether it already holds an item, picked from another cell, or not. The *pick* and *drop* operations are driven by corresponding probability functions.

Let C be the number of predefined classes, and $c = 1 \ldots C$ the class of a given item. The probability with which an agent picks an item of a given class c from the cell where it is currently located, referred to as

Table 1
Experimental results with monodimensional partitioning. Time is measured in ms.

N	w	T_{comp}	T_{comp}	T_{step}	α	β
4	w_l	89.78	90.79	174.84	0.001197	0.01452
4	w_m	122.58	120.54	234.2	0.001225	0.01446
4	w_h	151.40	149.96	287.36	0.001211	0.01439
9	w_l	39.69	89.38	147.44	0.001190	0.01430
9	w_m	53.44	121.27	202.38	0.001202	0.01455
9	w_h	67.74	147.73	259.80	0.001219	0.01418

Table 2
Experimental results with bidimensional partitioning. Time is measured in ms.

N	w	T_{comp}	T_{adj}	T_{diag}	T_{step}	α	β
4	w_l	88.65	45.63	4.40	143.01	0.00118	0.01460
4	w_m	119.09	60.45	5.49	183.43	0.00119	0.01450
4	w_h	149.31	76.58	6.58	236.53	0.00119	0.01470
9	w_l	40.48	30.99	4.30	83.93	0.00121	0.01487
9	w_m	55.05	41.11	5.23	120.62	0.00123	0.01480
9	w_h	67.38	50.66	5.90	147.58	0.00121	0.01459

$P_{pick}(c)$, is defined as:

$$P_{pick}(c) = \left(\frac{k_p}{k_p + f(c)}\right)^2 \tag{6}$$

In formula (6), $f(c)$ gives the number of items of class c, accumulated in the cells within the visibility area of the agent, divided by the overall number of items of *all* classes that are located in the same area. As more items of a class c are accumulated in the visibility area of the agent, $f(c)$ increases and the value of the pick probability for this class becomes lower, and vice versa. This has the effect of inducing agents to pick items that are uncommon in the visibility area, and ignore items of the class that is being accumulated. The parameter k_p is a non-negative value used to tune the clustering effort. In the tests performed in this work it is set to 0.1, as in (Deneubourg et al., 1990).

The probability that a loaded agent *drops* an item of class c, $P_{drop}(c)$, is defined as:

$$P_{drop}(c) = \left(\frac{f(c)}{k_d + f(c)}\right)^2 \tag{7}$$

The drop probability increases as more items of class c are accumulated in the visibility area of the agent. In this work, the parameter k_d is set to 0.3, as in (Deneubourg et al., 1990).

The effectiveness of the sorting algorithm is evaluated through a spatial entropy function, based on the well-known Shannon formula for the calculation of information content. The value of the entropy measures how well items are sorted in the territory. At the beginning of the experiments items of the different classes are uniformly spread over the territory, and the entropy value is close to 1. Then the value decreases to values close to 0, meaning that the items have been sorted. The experiment ends when the entropy value gets stabilized. More details on this aspect can be found in (Bonabeau et al., 1999) and (Forestiero et al., 2008).

4.2. Performance results on the real parallel platform

The ant-based sorting algorithm has been executed on a cluster in which each computing node has CPU Intel(R) Xeon(R) CPU E5-2670 2.60 GHz and 128 GB RAM. As agent-based infrastructure, we used the `APSwarm` platform (Cicirelli et al., 2016a). The nodes are interconnected with an Intel Corporation I350 Gigabit Network. The territory on which the ants operate is a square of 240×240 space units, so the area A is equal to 57,600 square units. In the different sets of experiments, the space was equally partitioned among four nodes and nine nodes, both with monodimensional partitioning and bidimensional partitioning. With bidimensional partitioning, the resulting regions are square-shaped: this choice is consistent with the result shown in Section 3.3, i.e., the square shape minimizes the communication burden. The execution of one step on a computing node (see Section 2) corresponds to the execution of one movement and of one pick or drop operation for all the ant agents that are located on the region assigned to the node.

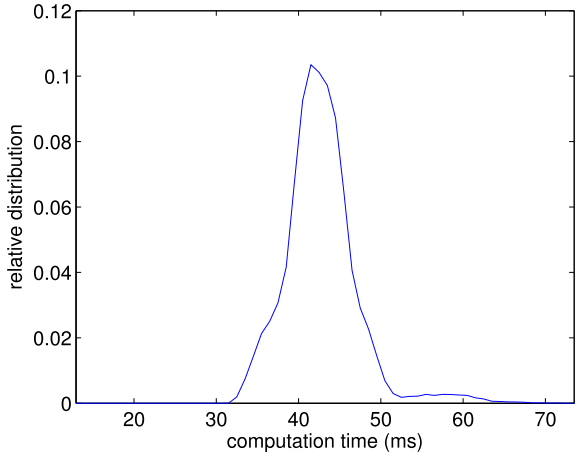
The overall number of agents is referred to as N_{agents}, and their density, $w = N_{agents}/A$, is uniform in the territory. We executed the algorithm with three different values of the workload: a "low" workload, corresponding to $N_{agents} = 300{,}000$, a "medium" workload, corresponding to $N_{agents} = 400{,}000$ agents, and a "high" workload, corresponding to $N_{agents} = 500{,}000$ agents. The three values of agent density that correspond to low, medium, and high workload are respectively denoted as w_l, w_m and w_h. The respective values are 5.21, 6.94 and 8.68 agents/square units. In all the three cases, the number of objects that need to be clustered by ant agents was set to twice the number of agents.

Table 1 shows the results of the experiments performed for the scenario of monodimensional partitioning. In particular, the table reports the average local computation time, referred to as T_{comp}, the average communication time, referred to as T_{comp}, the average step time T_{step} (defined at the beginning of Section 3), and the computation/communication densities, denoted as α and β, which will be defined shortly. Analogous results are reported in Table 2 for the case of bidimensional partitioning, except that the average communication time is separately computed for the case of regions that are adjacent horizontally or vertically,[5] and for the case of regions that are adjacent diagonally. The two quantities are denoted, respectively, as T_{adj} and T_{diag}. All the reported values are obtained by averaging the results of 10 runs: for each metric, we computed the 95% confidence intervals $\overline{X} \pm \lambda\overline{X}$, obtained with Student's t-distribution, where $\overline{X}$ is the mean value of the metric. The value of λ resulted to be always lower than 0.05, which is a good indication of the reliability of the results.

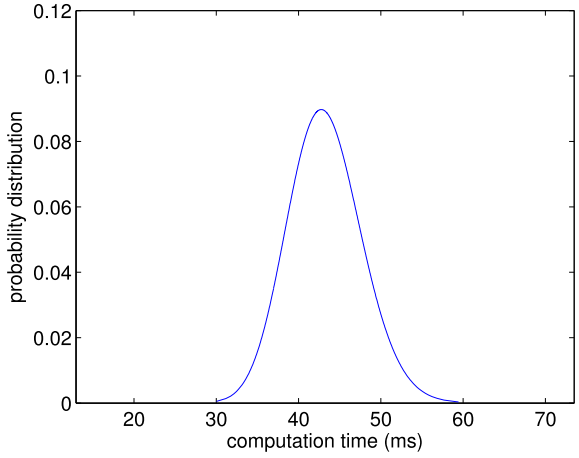
We define the computation density α as the amount of computation time per agent, and the communication density β as the amount of communication time per agent.[6] Both values, reported in Tables 1 and 2,

[5] Since we are considering square-shaped regions, north-south and east-west communications are equivalent.

[6] For the scenario of bidimensional partitioning, the value of β is defined with respect to the communication time T_{adj}, as this time is much larger than T_{diag} and therefore has a higher effect on the overall performance.



(a) Real frequency distribution



(b) Gamma distribution

Fig. 10. (a) real frequency distribution of the local computation time for the scenario of bidimensional partitioning with nine nodes and low load; (b) pdf of the Gamma distribution with same average and standard deviation as the real frequency distribution.

are almost constant, always within a 5% difference.

For the case of monodimensional partitioning, the average computation and communication times can be expressed with equations (8) and (9), respectively:

$$T_{comp} = \alpha \cdot w \cdot \frac{A}{N} \tag{8}$$

$$T_{comp} = \beta \cdot w \cdot rH \tag{9}$$

where A is the size of the whole territory, N is the number of nodes, r is the visibility radius of agents and H is the height of the space, as in Fig. 8. In the above expressions, the average computation and communication times are both expressed as the product of three factors related, respectively, to: (i) the computation and communication densities, i.e., α and β; (ii) the agent density w, and (iii) the involved areas, i.e., A/N and rH.

For the case of bidimensional partitioning, the average computation time can be expressed as:

$$T_{comp} = \alpha \cdot w \cdot \frac{A}{N} = \alpha \cdot w \cdot \frac{H^2}{N} \tag{10}$$

The average communication times defined before, T_{adj} and T_{diag}, can be expressed as:

$$T_{adj} = \beta \cdot w \cdot \frac{r \cdot \sqrt{A}}{\sqrt{N}} = \beta \cdot w \cdot \frac{r \cdot H}{\sqrt{N}} \tag{11}$$

$$T_{diag} = \beta \cdot w \cdot r^2 \tag{12}$$

As for the case of monodimensional partitioning, the average computation and communication times are both expressed as the product of three factors related to the computation/communication density, the agent density and the involved area. The area involved in the expression of T_{diag}, r,[2] is considerably smaller than the area involved in the expression of T_{adj}, $(r \cdot H)/(\sqrt{N})$ since $r \ll H/\sqrt{N}$. As a result, $T_{diag} \ll T_{adj}$.

The expressions (8)–(11), and the fact that α and β are almost constant, allow us to estimate the values of T_{comp} and T_{adj} in different scenarios, for example with different numbers of nodes and different values of the visibility radius, both with monodimensional and bidimensional partitioning. These values are used in the framework to characterize the random variables of the transitions of the Petri net illustrated in Section 3.2, and then to predict the performance of the system in different scenarios. Prior to the performance analysis, illustrated in Section 4.4, in the next subsection we validate the presented Petri net model with respect to the real platform.

4.3. Validation of the petri net model

As discussed in the previous subsection, it is possible to estimate the values of the computation and communication times starting from the areas of the involved regions and borders. Such values can then be used in the execution of a Petri net that models the parallel computation process, as described in Section 3. The purpose is to estimate the average step time and then the global computation time of the parallel process. We first used the well-known Petri net tool Yasper (van Hee et al., 2006), specifically its “automatic simulation” tool, and then we wrote an ad hoc simulator in Matlab, which reproduces the same computation modeled by the Petri net, specifically the recursive expression (1). After verifying that the results are identical, we chose to execute the experiments with the Matlab simulator, which is faster and more flexible, as it permits to execute a parameter sweep analysis and to use any kind of distribution for computation and communication times, including those observed experimentally.[7]

We validated the simulation results with respect to the real platform in two phases: (i) we extracted the frequency distributions of the real computation and communication times, and used these distributions in the Matlab simulator to characterize the firing times of the Petri net transitions; (ii) we verified that the average step time obtained with simulation is very close to the average step time measured on the real platform.

Regarding the first phase, we collected the values of computation and communication times measured on the real platform and analyzed their frequency distributions. As an example, Fig. 10(a) shows the relative frequency histogram of the local computation time for the scenario with nine nodes, bidimensional space partitioning and low load. We computed the value of the Pearson coefficient to measure the correlation between the experienced frequency distribution and some well-known distributions, among which the normal distribution, the Gamma distribution, the Poisson distribution and the lognormal distribution. We found that the best similarity is with the Gamma distribution. In Fig. 10(b) we plot the probability density function of the Gamma distribution with the same values of average and standard deviation as

[7] Both the Petri net model and the Matlab simulator are available at the URL http://apswarm.icar.cnr.it/applications/j2ee-modules/sc/jnca.

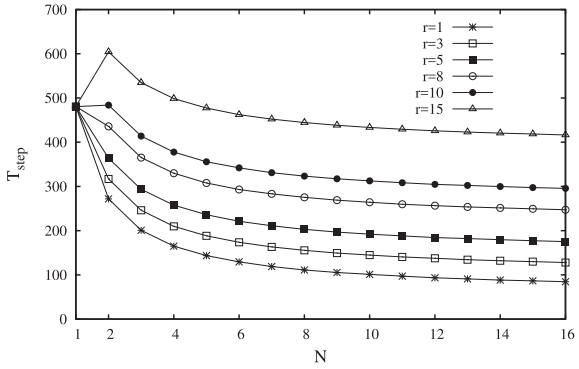


Fig. 11. Values of T_{step} in the case of monodimensional partitioning, vs. the number of nodes N, for different values of r.

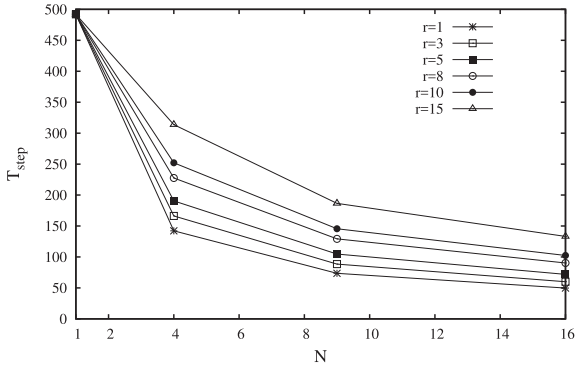


Fig. 12. Values of T_{step} in the case of bidimensional partitioning, vs. the number of nodes, for different values of r.

the real distribution.[8] It appears that the Gamma distribution with the shape parameter equal to 25 is a good approximation of the real distribution, and indeed the Pearson coefficient is equal to 0.981. A similar conclusion was drawn for the communication time.

For the second phase of the validation, we run the Matlab simulator for all the scenarios described before and summarized in Tables 1 and 2. To characterize the computation and communication times used in the simulator, we used the Gamma distribution as described before. We found that the values of the average step time computed with Matlab never differ by more than 1.0% from the values obtained with the real experiments. This outcome ensures that the simulator has a very good reliability and also validates the use of the Gamma distribution to reproduce the real computation and communication times.

This validation allowed us to use the Matlab simulator to analyze different and more general scenarios, which is the topic of the next section.

4.4. Simulation analysis

The aim of this section is to analyze the performance of the parallel execution of the ant-based sorting algorithm, described in Section 4.1, using the Matlab simulator mentioned in Section 4.3. Two sets of results are illustrated in this section. First, we analyze the execution performance when varying the partitioning schema and the visibility radius. Then we discuss how the performance is related to the ratio between computation and communication load.

For the first set of results, we adopted the values of computation and communication densities defined and measured in Section 4.2, i.e., $\alpha = 0.0012$ and $\beta = 0.0143$. As explained in Section 4.2, these values were used to compute the values of computation and communication times, using expressions (8)–(12). In turn, the latter values were used in the Matlab simulator as the average values for the Gamma probability distributions of computation and communication times.

The simulation was executed for the same territory as the one described in Section 4.2, i.e., composed of 240×240 cells. The number of agents was set to 400,000, corresponding to the "medium" load defined in Section 4.2. We then varied the number of nodes among which the territory is partitioned and the visibility radius r, both with monodimensional and bidimensional partitioning. Fig. 11 shows the value of the average step time T_{step} vs. the number of nodes, in the case of monodimensional partitioning, for different values of the visibility radius. Many interesting conclusions can be drawn from these results. First, it is seen that the *sequential time*, i.e., the average step time experienced when all the computation is executed at a single node, is equal to about 480 time units: therefore, parallelization is convenient only for the values of r and N for which the average step time is lower than 480. For example, when the visibility radius is equal to 15, parallelization on four or less than four nodes is not convenient, meaning that the communication and synchronization overhead unmakes the advantage deriving from the computation partitioning. Parallelization begins to be convenient with five nodes or more. As the value of r reduces, parallelization becomes more and more effective: for example, with $r = 1$ and 16 nodes the average step time is about 85 time units.

Fig. 12 reports the values of T_{step} obtained in the case of bidimensional partitioning, with different values of N and r. In this case, parallelization is convenient in all the considered scenarios. Moreover, the value of T_{step} is always lower than the corresponding value obtained with monodimensional partitioning. For example, with $N = 16$ and $r = 1$, the value of T_{step} is about 50 time units, while it is equal to 85 time units with monodimensional partitioning. The reason is that the communication time is lower in the case of bidimensional partitioning: from Expressions (9) and (11) it is seen that the communication time T_{adj} is lower than the communication time experienced with monodimensional partitioning by a factor $\sqrt{N}$. The advantage of bidimensional partitioning proves that the lower communication time has a higher effect on overall performances than the larger number of nodes involved in synchronization barriers.[9] This result is interesting because it is hard to predict without resorting to experiments.

An interesting outcome of simulation experiments is that the average step time is strongly related to the ratio between the average communication time and the average computation time. Therefore, in the second set of experiments we analyzed this aspect.

In the case of monodimensional partitioning, the ratio T_{comp}/T_{comp}, from expressions (8) and (9), is:

$$\frac{T_{comp}}{T_{comp}} = \frac{\beta}{\alpha} \cdot \frac{r}{L} \cdot N \tag{13}$$

This ratio is equal to the product of three components: (i) the ratio between the communication and computation densities; (ii) a component with a geometrical meaning, i.e., the ratio between the visibility radius and the length of the territory, and (iii) the number of nodes among which the computation is distributed. Expression (13) can be formulated so as to isolate the first two contributions from the contribution related to the number of nodes:

$$\frac{T_{comp}}{T_{comp}} = R_l \cdot N \tag{14}$$

[8] The *shape* parameter of the Gamma distribution can be computed as μ^2/σ^2, and the *scale* parameter as σ^2/μ, where μ is the average and σ is the standard deviation.

[9] Synchronization barriers involve nine nodes with bidimensional partitioning and only three nodes with monodimensional partitioning.

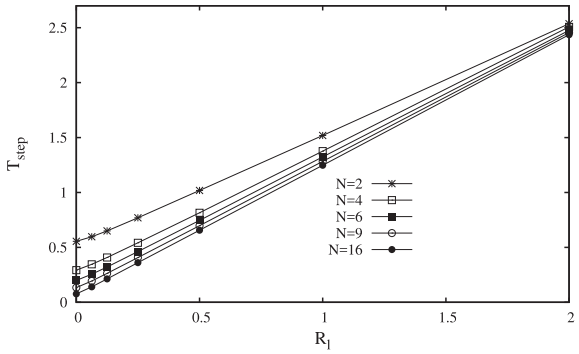


Fig. 13. Values of T_{step} vs. R_l, for different numbers of nodes, with monodimensional partitioning.

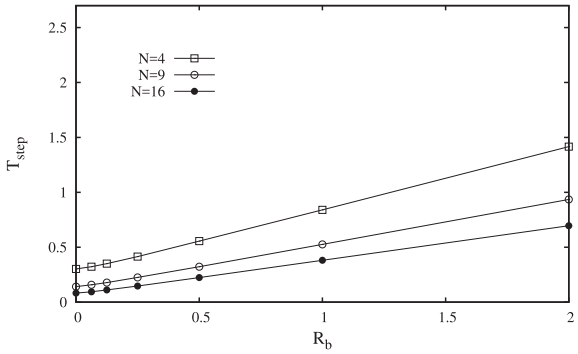


Fig. 14. Values of T_{step} vs. R_b, for different numbers of nodes, with bidimensional partitioning.

In Expression (14), the ratio R_l summarizes the relative weight of the communication with respect to computation, irrespective of the number of nodes. Fig. 13 reports the values of T_{step} obtained when varying the values of R_l and the number of nodes N. In these experiments, the *sequential time* is set to 1 so as to normalize all the other times with respect to it. Therefore, parallelization is convenient for the values of R_l and N for which the average step time is lower than 1. Fig. 13 shows that the average step time increases with the value of R_l, i.e., with the relative weight of the communication with respect to computation. It is also interesting to notice that the use of a larger number of parallel nodes is highly beneficial only when the relative weight of communication is low. For example, with $R_l = 0.125$, the average step time is 0.65 when using two nodes, while it is only 0.21 when using sixteen nodes. Conversely, when the weight of communication is high, e.g., with $R_l = 2$, the average step time is barely affected when increasing the number of nodes. The reason is that the communication time, which in this case prevails on the computation time, does not depend on the number of nodes, as seen in Expression (9). In conclusion, it is possible to take full benefit from parallelization, with monodimensional partitioning, only when the overhead of communication is limited.

In the case of bidimensional space partitioning, we focus on the communication time T_{adj} – since $T_{diag} \ll T_{adj}$, as discussed in Section 4.2 – and therefore on the ratio T_{adj}/T_{comp}. From expressions (10) and (11), for a square-shaped territory in which $H = L$, this ratio can be expressed as:

$$\frac{T_{adj}}{T_{comp}} = \frac{\beta}{\alpha} \cdot \frac{r}{L} \cdot \sqrt{N} \tag{15}$$

or, analogously to the monodimensional case, as:

$$\frac{T_{adj}}{T_{comp}} = R_b \cdot \sqrt{N} \tag{16}$$

This expression is very similar to Expression (14), and it is also noticed that the component that does not depend on the number of nodes, R_b, has the same expression as the analogous component R_l derived for the monodimensional scenario. However, this time the ratio increases more slowly with the number of nodes, as it is proportional to $\sqrt{N}$ instead of N. Fig. 14 shows how the value of T_{step} varies with the values of R_b and of the number of nodes N. As opposed to the case of monodimensional partitioning, increasing the number of nodes allows to improve the performance, i.e., to reduce the value of T_{step}, for each value of the ratio R_b. Indeed, as seen in Expression (11), the communication time decreases when the number of nodes increases. Therefore there is a benefit from a higher degree of parallelization due to the lower overhead related to the communication time. As discussed before, this benefit occurs despite the fact that each node must synchronize with eight neighbor nodes instead of only two, as in the monodimensional partitioning scenario.

Fig. 15 compares the values of T_{step} obtained with monodimensional and bidimensional partitioning. As we derived that the expressions of R_l and R_b are the same, it is possible to do this comparison by setting $R = R_l = R_b$. We see that the bidimensional partitioning is convenient for any value of R and N.

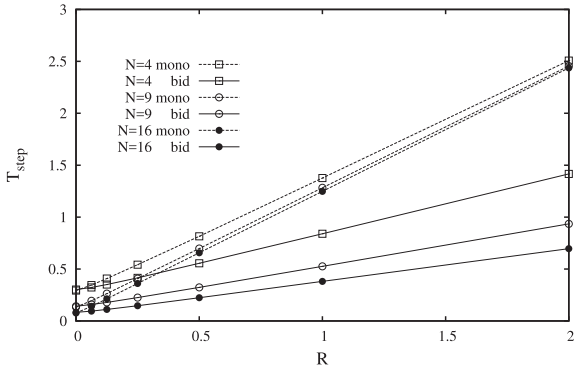


Fig. 15. Values of T_{step} vs. R, for different numbers of nodes. Comparison between monodimensional and bidimensional partitioning.

The described approach can be followed also with other types of space-aware application, to help the application experts individuate the most efficient option when both partitioning schemas – monodimensional and bidimensional – are admissible. More in particular, the approach consists in executing a set of tests on a real platform to derive the random distributions of computation and communication times. Then, these distributions can be used in the Matlab simulator to predict the performance with different space partitioning schemas.

5. Related work

The state-of-the-art in the recent literature clearly shows a growing interest on addressing the issues related to space-aware applications. In particular, agent-based spatially explicit models have been widely used to study a variety of phenomena including, for example, bio-inspired systems, traffic management and simulation, platforms for urban computing, GIS systems, etc. Multi-agent system (MAS) platforms provide support to execute and/or simulate systems modeled by using the agent metaphor. The services provided by MAS platforms include the support to agent lifecycle, communication among agents, agent perception and environment management. Especially for the case of spatially-explicit agent-based systems (Amouroux et al., 2007; Shook et al., 2013), the way the environment is represented, and the services made available to sense and act upon the environment are of utmost importance. Examples of well-known platforms where spatial environments are natively supported are Repast Simphony (North et al., 2013), Mason (Luke et al., 2005), NetLogo (Tisue and Wilensky, 2004) and Gama (Amouroux et al., 2007).

As the size and the sophistication of the problems increase, the exploitation of parallel and high-performance computing becomes mandatory (Shook et al., 2013; Parker and Epstein, 2011). If compared to the large number of existing MAS platforms, though, only few Parallel and Distributed MAS (PDMAS) platforms provide a native support for the parallel execution of agent-based systems. Moreover, the development of distributed systems based on PDMAS is a non-trivial task as it requires deep parallel programming skills, and the efficient and high performing execution of a model in a distributed context remains a challenging issue (Rousset et al., 2016; Cicirelli et al., 2015). A research effort for the automatic parallelization of general MAS systems is reported in (Scheutz and Schermerhorn, 2006), whereas some efforts related to the automatic parallelization of spatially-explicit agent-based systems are reported in (Cicirelli et al., 2015, 2016a; Scheutz and Schermerhorn, 2006).

Examples of PDMAS platforms suited to execute spatially-explicit agent-based systems are D-MASON (Cordasco et al., 2013), Flame (Coakley et al., 2012), Pandora (Angelotti et al., 2001), RepastHPC (Collier and North, 2012) and Swages (Scheutz et al., 2006). These platforms differ from each other on the basis of features like the adopted programming language, the mechanisms used to support distributed communications, the exploited synchronization mechanisms among computing nodes, the support to load balancing and so forth (Rousset et al., 2016). In D-MASON (Cordasco et al., 2013) and RepastHPC (Collier and North, 2012), the environment is divided into cells, which are assigned to different computing nodes in order to distribute the developed system. Both a monodimensional and a bidimensional cell partitioning are supported. To provide the perception of a continuous environment, even in a distributed setting, overlapping zones, also called “Area of Interest” (AOI), are considered. Managing AOIs require to make local copies of the *borders* among adjacent partitions, and keep such copies updated. The concept of AOI derives from the so-called *ghost areas* originally introduced in (Ripeanu et al., 2001). The main difference between the D-MASON and RepastHPC is that in the former work the notion of cells and partitions coincide, while in the latter the cells are of equal size and a partition is composed of a subset of contiguous cells. Also Pandora (Angelotti et al., 2001) exploits an approach similar to D-MASON and RepastHPC for system distribution.

In Flame (Coakley et al., 2012), a MAS system is distributed over the computing nodes on the basis of the agents’ spatial coordinates that can be either in 2D or 3D. The environment is still perceived as continuous but it is partitioned by considering agents’ distribution. Agents that can potentially communicate among each other are grouped on the same node with the aim of optimizing the overhead generated by inter-node communications.

A different approach for managing the environment relies on the use of tuple-based middlewares. The TOTA middleware (Mamei and Zambonelli, 2004) exploits spatially distributed tuples, which are propagated across a network in order to implement diffusive spaces. Tuples can be injected into the system from any network node and can propagate and diffuse according to some tuple-specific propagation patterns. Agents can sense such tuples and decide how to react to them.

The high-performance parallel execution of computationally intensive and spatially-explicit agent-based systems on multicore platforms is considered in (Gong et al., 2013) and (Tang et al., 2011). The environment is decomposed into equal-sized horizontal regions and each region is assigned for execution on a different CPU core. In (Gong et al., 2013) the focus is on the spatial interactions among agents, on the spatial diffusion of opinions and on the consensus building that affects individual agent decisions. Execution performances are assessed by varying (i) the size and the population of the environment, and (ii) the range of interactions, i.e., the spatial distance over which agents are able to interact with other agents and with the environment. Mutual exclusion algorithms are used to regulate inter-region agent interactions thus avoiding conflicts among agents operating on different regions. In (Tang et al., 2011), instead, ghost zones are used to mirror areas of interest among partitions, and barrier and lock techniques are used for the synchronization of system execution among parallel processors.

In order to speed up the distributed execution of spatially-explicit agent-based systems while avoiding conflicts among agents, in (Rubio–Campillo and Cela, 2010) and (Wittek and Rubio-Campillo, 2012) each partition allocated for execution on a computing node is further divided into four matrices, each of them being managed by an independent logical process numbered from 0 to 3. The PDMAS only allows the execution of simultaneous logical processes in the distributed context if they have the same number, thus avoiding the concurrent execution of adjacent agents capable of modifying the same spatial partition. A similar, but more general, approach for solving conflicts was independently proposed in (Cicirelli et al., 2015) and (Cicirelli et al., 2016a). Here, a particular notion of logical time was purposely introduced in order to guarantee a conflict-free execution of agents allocated on different regions.

A framework for the efficient parallel execution of spatially-explicit agent-based systems is proposed in (Shook et al., 2013). The framework supports agent-grouping primitives, four different space partitioning techniques (monodimensional, bidimensional, quad-tree and recursive bisection partitioning), a communication-aware load balancing strategy, and the use of proxy (ghost) entities as local copies of entities residing on other partitions. Agent groups and space partitions are then assigned for execution on the available computing nodes. The main goal is to reduce inter-process communication that, as confirmed by simulation experiments, heavily impairs the achievable execution performance.

Recently, the emerging of Cloud platforms has increased the possibility of improving the execution of parallel algorithms, for example by resorting to a large number of servers on a single or on multiple data centers. The relationship between the amount of computation and the amount of communication is recognized as an important issue when executing parallel Cloud applications. The transmission of data among distributed nodes of a Cloud platform heavily influences the scalability and the performance of parallel applications (ExpóSito et al., 2013). Therefore, the estimation of the amount of data that needs to be transmitted among the computing nodes is of paramount importance. In (Roloff et al., 2012), different communication patterns are evaluated, and the authors notice that none of the major Cloud providers offer information about the specific network interconnections used among the machines, and it is impossible to determine the network latency or distance between nodes. This can provide challenges to communication-intensive applications, and it is a major disadvantage compared to on-premises clusters. A possible solution to this lack of control can be addressed by the SDN (Software-Defined Networking) paradigm, which allows routing and forwarding rules to be controlled by software instead of being hardwired in the routers (Malik et al., 2013).

6. Conclusion and future work

This paper has presented an original framework aimed at assessing and predicting the execution performance of agent-based space-aware applications (SAAs) that are executed in a parallel context. The parallelization is obtained by partitioning the territory into regions and assigning each region to a distinct computing node. The presented framework relies on Stochastic Time Petri Nets, and it is able to evaluate the performance of a parallel SAA for different execution settings. An execution setting is characterized by the type of territory partitioning (i.e., monodimensional or bidimensional), the number of parallel computing nodes, and the amount of computation assigned to the agents. The approach allows to predict the performance starting from the knowledge of the random distributions of local computation and communication times. We validated the framework, for the well-known ant clustering and sorting algorithm, by comparing the performance experienced in a real platform to that predicted by the framework. We also performed an extensive set of simulation experiments to assess the

performance in more general scenarios. An outcome of the experiments is that the performance is strongly related to the ratio between local computation and communication load. The expressions derived for this ratio allowed us to compare the performance obtained when adopting, for the same SAA, monodimensional and bidimensional territory partitioning. From the experiments, it emerged that the bidimensional partitioning performs better despite its higher synchronization cost.

The presented approach can be readily exploited when the spatial distribution of agents is uniform. When this assumption does not hold, the actual spatial distribution of the load must be considered to obtain the random distributions of computation and communication times for the different regions, and then use these distributions in the simulator. In addition, the approach must be extended when there is the need to include "global" operations, i.e., operations that concern the whole territory and/or all the agents.

Prosecution of the work is geared at using the framework to asses the performance of SAAs where: (i) the local computation and communication times are characterized by different random distributions; (ii) the agents and the computation are not uniformly distributed; (iii) different techniques for territory partitioning are adopted, e.g., quad-tree or recursive bisectional. We are also building an analytical model of the system by using a Markov chain, for which we define a state for each possible marking of the Petri net, and compute the transition rate between each couple of states. This analysis can be used, for example, to compute the average time that each node needs to wait at a synchronization barrier.

Acknowledgements

This work was partially funded by the project DOMUS, MIUR grant PON03PE_00050_2.

Appendix A. Stochastic Time Petri Nets

A Stochastic Time Petri Net (STPN) is a tuple[10]:
$\langle P, T, A^-, A^+, C, F \rangle$
where P and T are disjoint sets of places and transitions, $A^- \subseteq P \times T$ and $A^+ \subseteq T \times P$ define arc connections, C is a set of cumulative distribution function, and $F : T \to C$ associates a cumulative distribution function to each transition.

Given an STPN $\langle P, T, A^-, A^+, C, F \rangle$, a marking $m \in \mathbb{N}^P$ assigns a natural number of tokens to each place of the net. A transition t is *enabled* by m if m assigns at least one token to each of its input places. The set of transitions enabled by m is denoted as $E(m)$.

Let $c_i = \langle m_i, \overrightarrow{\tau_i} \rangle$ be a couple in which m is a marking and the vector $\overrightarrow{\tau_i} \in \mathbb{R}_{\geq 0}^{E(m)}$ assigns a *firing time* to each enabled transition, and let $c_0 = \langle m_0, \overrightarrow{\tau_0} \rangle$ be the couple composed of the initial marking m_0 and the initial vector $\overrightarrow{\tau_0}$ of firing times. An execution of the STPN is given by a (finite or infinite) path: $c_0 \xrightarrow{\gamma_1} c_1 \xrightarrow{\gamma_2} c_2 \xrightarrow{\gamma_3} c_3 \ldots$
where $\gamma_i \in T$ is the ith transition fired along the execution of the net. After the firing of γ_i:

- the next transition γ_{i+1} is selected from the set of enabled transitions with the minimum time to fire;
- after the firing of γ_{i+1}, the new marking m_{i+1} is derived by (i) removing a token from each place p such that $(p, \gamma_{i+1}) \in A^-$, (ii) adding a token in each place p such that $(\gamma_{i+1}, t) \in A^+$. A transition t enabled by m_{i+1} is said to be *persistent* if it is distinct from γ_{i+1}, and it is enabled also by m_i and by the intermediate marking after steps (i) and (ii); otherwise t is said to be *newly enabled*;
- for each newly enabled transition t, its time to fire is sampled according to the distribution $F(t)$; for each persistent transition t, the time to fire of t is reduced by the sojourn time in the previous marking, i.e., $\vec{\tau}_{i+1}(t) = \vec{\tau}_i(t) - \vec{\tau}_{i+1}(\gamma_{i+1})$.

References

Amouroux, E., Chu, T.-Q., Boucher, A., Drogoul, A., 2007. Gama: an environment for implementing and running spatially explicit multi-agent simulations. In: Pacific Rim International Conference on Multi-agents. Springer, pp. 359–371.

Angelotti, E.S., Scalabrin, E.E., Ávila, B.C., 2001. Pandora: a multi-agent system using paraconsistent logic. In: Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings. Fourth International Conference on. IEEE, pp. 352–356.

Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: a survey. Comput. Netw. 54 (15), 2787–2805.

Bonabeau, E., Dorigo, M., Theraulaz, G., 1999. Swarm Intelligence: from Natural to Artificial Systems. Oxford University Press, New York, NY, USA.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. ACM, pp. 13–16.

Cicirelli, F., Giordano, A., Nigro, L., 2015. Efficient environment management for distributed simulation of large-scale situated multi-agent systems. Concurrency Comput. Pract. Ex. 27 (3), 610–632.

Cicirelli, F., Forestiero, A., Giordano, A., Mastroianni, C., 2016. Transparent and efficient parallelization of swarm algorithms. ACM Trans. Autonom. Adapt. Syst. 11 (2) 14:1–14:26.

Cicirelli, F., Forestiero, A., Giordano, A., Mastroianni, C., Spezzano, G., 2016. Parallel execution of space-aware applications in a cloud environment. In: 24th Euromicro Int. Conf. On Parallel, Distributed, and Network-based Processing, PDP 2016, Heraklion, Greece, pp. 686–693.

Coakley, S., Gheorghe, M., Holcombe, M., Chin, S., Worth, D., Greenough, C., 2012. Exploitation of high performance computing in the flame agent-based simulation framework. In: High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on. IEEE, pp. 538–545.

Collier, N., North, M., 2012. Repast HPC: a Platform for Large-scale Agentbased Modeling. Wiley.

Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., Spagnuolo, C., 2013. Bringing together efficiency and effectiveness in distributed simulations: the experience with d-mason. Simulation 89 (10), 1236–1253.

Cosenza, B., Cordasco, G., Chiara, R.D., Scarano, V., 2011. Distributed load balancing for parallel agent-based simulations. In: 19th Euromicro Int. Conf. On Parallel, Distributed, and Network-based Processing, PDP 2011, Ayia Napa, Cyprus, pp. 62–69.

Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., Chrétien, L., 1990. The dynamics of collective sorting robot-like ants and ant-like robots. In: Proc. Of the First International Conference on Simulation of Adaptive Behavior on from Animals to Animats. MIT Press, Cambridge, MA, USA, pp. 356–363.

Ekanayake, J., Fox, G., 2010. High performance parallel computing with clouds and cloud technologies. In: Cloud Computing. Springer, pp. 20–38.

ExpóSito, R.R., Taboada, G.L., Ramos, S., Touriño, J., Doallo, R., 2013. Performance analysis of hpc applications in the cloud. Future Generat. Comput. Syst. 29 (1), 218–229.

Forestiero, A., Mastroianni, C., Spezzano, G., 2008. Qos-based dissemination of content in grids. Future Generat. Comput. Syst. 24 (3), 235–244.

Garofalo, G., Giordano, A., Piro, P., Spezzano, G., Vinci, A., 2017. A distributed real-time approach for mitigating CSO and flooding in urban drainage systems. J. Netw. Comput. Appl. 78, 30–42.

Gong, Z., Tang, W., Bennett, D.A., Thill, J.-C., 2013. Parallel agent-based simulation of individual-level spatial interactions within a multicore computing environment. Int. J. Geogr. Inf. Sci. 27 (6), 1152–1170.

Harri, J., Filali, F., Bonnet, C., 2009. Mobility models for vehicular ad hoc networks: a survey and taxonomy. IEEE Commun. Surv. Tutor. 11 (4), 19–41.

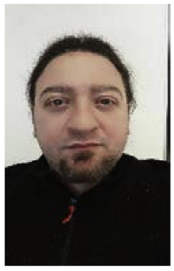
Hu, P., Dhelim, S., Ning, H., Qiu, T., 2017. Survey on fog computing: architecture, key technologies, applications and open issues. J. Netw. Comput. Appl. 98, 27–42.

[10] In this paper, only a subset of the features of the STPNs are exploited. In particular, with reference to the definitions given in (Paolieri et al., 2016), for each transition we assume: an *enabling function* that evaluates always to *true*; an *earliest firing time* and a *latest firing time* of 0 and $+\infty$, respectively; a constant *weight function*; a null *update function*.

Krishnan, Y.N., Bhagwat, C.N., Utpat, A.P., 2015. Fog computing- network based cloud computing. In: Electronics and Communication Systems (ICECS), 2015 2nd International Conference on. IEEE, pp. 250–251.
I. Lee, K. Lee, The Internet of Things (IoT): Applications, Investments, and Challenges for Enterprises, Business Horizons.
Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G., Mason, 2005. A multiagent simulation environment. Simulation 81 (7), 517–527.
Malik, M.S., Montanari, M., Huh, J.H., Bobba, R.B., Campbell, R.H., June 24-27, 2013. Towards SDN enabled network control delegation in clouds. In: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, pp. 1–6.
Mamei, M., Zambonelli, F., 2004. Programming pervasive and mobile computing applications with the tota middleware. In: Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on. IEEE, pp. 263–273.
Murata, T., 1989. Petri nets: properties, analysis and applications. Proc. IEEE 77 (4), 541–580.
North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M., Sydelko, P., 2013. Complex adaptive systems modeling with repast simphony. Complex Adapt. Syst. Model. 1 (1), 1.
Paolieri, M., Horváth, A., Vicario, E., 2016. Probabilistic model checking of regenerative concurrent systems. IEEE Trans. Software Eng. 42 (2), 153–169.
Parker, J., Epstein, J.M., 2011. A distributed platform for global-scale agent-based models of disease transmission. ACM Trans. Model Comput. Simulat 22 (1), 2.
Peterson, J.L., 1977. Petri nets. ACM Comput. Surv. 9 (3), 223–252.
Ripeanu, M., Iamnitchi, A., Foster, I., 2001. Cactus application: performance predictions in grid environments. In: European Conference on Parallel Processing. Springer, pp. 807–816.
Roloff, E., Diener, M., Carissimi, A., Navaux, P.O.A., 2012. High performance computing in the cloud: deployment, performance and cost efficiency. In: Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, pp. 371–378.
Rousset, A., Herrmann, B., Lang, C., Philippe, L., 2016. A survey on parallel and distributed multi-agent systems for high performance computing simulations. Comput. Sci. Rev. 22, 27–46.
Rubio-Campillo, X., Cela, J., 2010. Large-scale agent-based simulation in archaeology: an approach using high-performance computing. In: Proceedings of the 38th Annual Conference on Computer Applications and Quantitative Methods in Archaeology, Granada, Spain, pp. 153–159.
Scheutz, M., Schermerhorn, P., 2006. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. J. Parallel Distr. Comput. 66 (8), 1037–1051.
Scheutz, M., Schermerhorn, P., Connaughton, R., Dingler, A., 2006. Swages-an extendable distributed experimentation system for large-scale agent-based alife simulations. Proceed. Artif. Life X, 412–419.
Shook, E., Wang, S., Tang, W., 2013. A communication-aware framework for parallel spatially explicit agent-based models. Int. J. Geogr. Inf. Sci. 27 (11), 2160–2181.
Tang, W., Bennett, D.A., Wang, S., 2011. A parallel agent-based model of land use opinions. J. Land Use Sci. 6 (2–3), 121–135.
Tisue, S., Wilensky, U., 2004. Netlogo: design and implementation of a multi-agent modeling environment. In: Proceedings of Agent, vol. 2004, pp. 7–9.
van Hee, K., Oanea, O., Post, R., Somers, L., an der Werf, J. M. v., 2006. Yasper: A tool for workflow modeling and analysis. In: Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD '06). IEEE Computer Society, Washington, DC, USA, pp. 279–282.
Wittek, P., Rubio-Campillo, X., 2012. Scalable agent-based modelling with cloud hpc resources for social simulations. In: Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, pp. 355–362.
Wooldridge, M., 2002. An Introduction to Multi-agent Systems. John Wiley & Sons, Ltd.

Franco Cicirelli, Ph.D, is a researcher at ICAR-CNR (Italy) since December 2016. He earned a Ph.D in System Engineering and Computer Science at the University of Calabria (Italy). He was a researcher fellow at the University of Calabria (Italy) from 2006 to 2015. His research work mainly focuses on Software Engineering tools and methodologies for the modeling, analysis and implementation of complex time-dependent systems. Research topics are agent-based systems, distributed simulation, parallel and distributed systems, real-time systems, workflow management systems, Internet of Things and cyber-physical systems. His research activities involve also Petri Nets, Timed Automata and the DEVS formalism.

Agostino Forestiero received the Laurea degree in computer engineering and the Ph.D. degree in computer engineering from the University of Calabria, Cosenza, Italy, in 2002 and 2007, respectively. He is a researcher at the CNR Institute for High Performance Computing and Networks, Rende, Italy. He has published or presented more than 40 scientific papers on international journals and conferences. His research interests include pervasive computing, cloud and fog computing, social mining and swarm intelligence. He has served as a Program Committee Member of several conferences. He is co-founder of the Eco4Cloud company (www.eco4cloud.com).



Andrea Giordano is a researcher at the National Research Council of Italy (CNR) – Institute for High Performance Computing and Networking (ICAR) - since March 2011. He earned a Ph.D in System Engineering and Computer Science at the University of Calabria (Italy), where he also earned a Master's degree in Computer Engineering. His research work mainly focuses on: agent-based systems, parallel and distributed systems, swarm intelligence, distributed simulation, Internet of things and cyberphysical systems.

Carlo Mastroianni received the Laurea degree and the PhD degree in computer engineering from the University of Calabria, Italy, in 1995 and 1999, respectively. He is a researcher at the Institute of High Performance Computing and Networking of the Italian National Research Council, ICAR-CNR, in Cosenza, Italy, since 2002. Previously, he worked at the Computer Department of the Prime Minister Office, in Rome. He co-authored more than 100 papers published in international journals, among which IEEE/ACM TON, IEEE TCC, IEEE TEVC and ACM TAAS, and conference proceedings. He edited special issues for the Journals Future Generation Computer Systems, the Journal of Network and Computer Applications, the Computer Networks Multiagent and Grid Systems. His areas of interest are cloud computing, urban computing, bio-inspired algorithms, multi-agent systems.