Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, eds.

PROGRAMMING AGENT-BASED DEMOGRAPHIC MODELS WITH CROSS-STATE AND MESSAGE-EXCHANGE DEPENDENCIES: A STUDY WITH SPECULATIVE PDES AND AUTOMATIC LOAD-SHARING

Alessandro Pellegrini
Francesco Quaglia

Department of Computer, Control,
and Management Engineering
Sapienza, University of Rome
Roma, ITALY

Cristina Montañola-Sales
Josep Casanovas-García

inLab FIB, Barcelona School of Informatics
Universitat Politcnica de Catalunya – BarcelonaTech
Barcelona Supercomputing Center
08034 Barcelona, SPAIN

ABSTRACT

Agent-based modeling and simulation is a versatile and promising methodology to capture complex interactions among entities and their surrounding environment. A great advantage is its ability to model phenomena at a macro scale by exploiting simpler descriptions at a micro level. It has been proven effective in many fields, and it is rapidly becoming a de-facto standard in the study of population dynamics. In this article we study programmability and performance aspects of the last-generation ROOT-Sim speculative PDES environment for multi/many-core shared-memory architectures. ROOT-Sim transparently offers a programming model where interactions can be based on both explicit message passing and in-place state accesses. We introduce programming guidelines for systematic exploitation of these facilities in agent-based simulations, and we study the effects on performance of an innovative load-sharing policy targeting these types of dependencies. An experimental assessment with synthetic and real-world applications is provided, to assess the validity of our proposal.

1 INTRODUCTION

Context. Agent-based modeling (ABM) is a simulation technique providing abstract representations of a scenario via a descriptive model targeted at reproducing its evolution through its components, including their decision-making capabilities and interaction patterns. An agent, which is a component of the overall descriptive model, can be defined as an entity (theoretical, virtual, or physical) capable of acting on itself, on the environment in which it evolves, and capable of communicating/interacting with other agents Jennings et al. (1998). ABM is very effective in capturing interactions at a macro scale which directly or indirectly come from the way agents behave at a micro scale level. In this sense, the individual or collective interactions among agents can be used to effectively derive properties of general systems which could be difficult to study by using more traditional simulation techniques. Therefore, the intrinsic expressive power offered by ABM makes it a proven solution to explore complex real-world scenarios, such as disaster rescue Takahashi et al. (2002), ancient societies resilience Balbo et al. (2014), epidemiology Prats et al. (2016), and economic analysis Page (2008).

From a modeling point of view, the main advantage of adopting ABM comes from that individual-specific explanatory variables can be included in the model, such as age, education level, salary group, and ethnicity to model the number of children that an individual female will have. This capability has attracted quantitative social science researchers and practitioners such as anthropologists, historians, and demographers to investigate the potential use of micro-level simulation models in their research.

From a technical point of view, supporting the execution of simulation models expressed using such a versatile solution is a task which requires a substantial methodological effort. In fact, a large number of widely-adopted ABM frameworks Tisue and Wilensky (2004), Minar et al. (1996), Luke et al. (2005), North et al. (2005) is intrinsically serial, and can therefore handle a *population* which is significantly limited in its size—on the order of thousands. To avoid limiting the speed and scalability of simulations, which are becoming always more demanding in terms of computing resources, efficient parallelization techniques must be employed. On this trend, several works aim at exploiting the high parallelism offered by GPU computing Lysenko and D'Souza (2008), Park and Han (2008) or cluster-based parallel computing **?**. From a more general point of view, Discrete Event Simulation (DES) can be considered as a mainstream formalism to describe agent-based models. The reason is that agents' interactions with other entities can be abstracted as occurring at specific time instants—interactions having a specific duration can be anyhow mapped to a couple of *begin* and *end* discrete events. The mapping from ABM to DES is quite trivial, as all the entities of an ABM (namely, agents and the environment) can be easily mapped to the more general notion of Logical Process (LP), proper of DES environments. This is an important aspect, given the existence of a plethora of techniques, globally referred to as Parallel Discrete Event Simulation (PDES) Fujimoto (1990), which provide protocols and mechanisms for running complex DES models in parallel. This allows for speedup in the model execution and tractability of highly-complex and/or large/huge models.

Target problem and background results. Nevertheless, ABM expresses a degree of interaction among the entities which could be orders of magnitude greater than conventional models run on top of PDES environments. Moreover, given that ABM comes from historically-sequential environments, its migration towards parallel or distributed environments (dictated by the larger scale of modern simulations) deserves a great care, especially if it is required that during the migration process no or minimal intervention is demanded from the modeler. This is an important aspect provided that, as discussed, in many cases modelers come from research fields which could possibly be unrelated with computer science. In particular, in ABM agents might be given the freedom to interact in a twofold way. On the one hand, they can communicate *explicitly*, namely via message passing, which is the traditional way to exchange information among different LPs in a PDES system. On the other hand, since ABM has been traditionally supported via sequential execution, nothing prevents agents to directly (in-place) modify the state of any other entity of the simulated system. In a sequential execution, this does not hamper the correctness of the model, as any event is always executed in a *causal-consistent way*, namely in non-decreasing timestamp order. When a simulation model is run in parallel, especially according to a speculative synchronization protocol Jefferson (1985), this could be no longer the case. This scenario might produce consistency issues, as it could be impossible for the underlying runtime environment to reconstruct a correct simulation state after a causal inconsistency is a-posteriori detected. In fact, the runtime environment might have no possibility to identify (at least at no or negligible cost) whether a LP is performing in-place accesses, thus making this form of coordination among LPs an *implicit* one.

The work in Pellegrini and Quaglia (2014b) provides the support for coping with this issue, integrated within the open source ROOT-Sim speculative PDES platform Pellegrini and Quaglia (2014a). In fact, it proposes a solution where the PDES environment offers a programming model which allows to implement simulation models in a purely sequential style, while enabling a fully-transparent deploy on top of parallel multi/many-core architectures. According to this programming model (and its associated runtime support), the simulation model's software is no longer limited to perform per-event state access/update operations in data separation across the LPs. Rather, state-wide in-place access by event handlers, exploiting shared memory in the underlying multi-core machine, is offered as a powerful means to improve programmability of complex models and further increasing model's execution speedup—this is the materialization of what is referred to as a *cross-state interaction*. As an example, it can avoid the coding of complex cross-LP event patterns in case some data embedded within the state of a given LP must be used while processing an event occurring at some other LP. Also, for (very) large models, where such data awareness would involve large

amounts of LPs, the possibility to directly access the target data in-place while coding the event handlers for whatever LP would also lead to avoiding large message-exchange overhead. This programming model perfectly embraces the ABM paradigm, in which, as an example, agents are allowed to gather in a certain region of space for a particular time span, and then migrate towards a different portion of the environment. These temporary gatherings can be coded in a more fruitful and easier way by implementing ad-hoc event handlers which directly (in-place) modify the content of whichever agent is currently part of the transient group. Moreover, this programming model allows for a completely-transparent deploy of sequential-style models on top of shared-memory parallel architectures.

However, this might not be sufficient to deliver high-performance execution of the simulations. In fact, the runtime dynamics of the model, in terms of events' granularity and volume of the interactions, might significantly affect the overall performance. To this end, another recent proposal Marziale et al. (2016) has tackled the issue of determining at runtime what could be the more valuable grouping of LPs with respect to in-place state accesses. We note that these groupings could be possibly (yet not necessarily) related to the temporary logical gathering of agents within a certain portion of the environment, and this can be detected without any manual intervention from the simulation-model developer. The work in Marziale et al. (2016) explicitly determines these groupings depending on the frequency of in-place accesses by the various LPs of the system, and exploits this information to bind at runtime a set of LPs (involved in a certain grouping/gathering) on a specific worker thread. By using this approach, the ultimate goal is to explicitly limit the optimism of the underlying speculative runtime engine, so that events associated with a certain group of LPs are executed in timestamp order—a given worker thread, in fact, executes events according to the Lowest Timestamp First (LTF) strategy considering all LPs bound to it.

One point not considered in Marziale et al. (2016) is the co-presence of message passing-based and direct in-place interactions. Returning to the ABM scenario, let us consider a simple example in which, at a certain execution phase, we can identify two different gatherings of agents, which repeatedly perform in-place accesses to each other's states. Simply grouping the LPs associated with these agents on two separate worker threads might produce an increase in the rollback frequency if the two groups, at the same time, exchange a high volume of events via traditional message passing between certain LPs belonging to the two groups. This scenario can be easily mapped to real-world scenarios of demographic models in which agents represent people who compose families in a given region of the world, but have relatives in different countries which are the target of explicit actions (e.g., money transfer). In this simple scenario, the binding policy presented in Marziale et al. (2016) could select a suboptimal allocation of the workload onto the computing resources.

Contributions. In this paper we address two issues. On the one hand, we clearly specify agent-based programming guidelines for demographic simulations to be hosted by last-generation PDES platforms coupling the dual (cross-state and message-exchange) interaction modes. This has the twofold benefit of allowing demographers to concentrate much more on modeling aspects rather than on parallelization ones, while at the same time enabling for reuse of previous models, which have possibly already gone through a thorough validation phase. On the other hand, we explicitly tackle the more complex interaction pattern (based on implicit and explicit synchronization) which is often exhibited by ABM, so as to propose a new analytic model to enforce load-sharing and fruitful usage of the resources offered by multi/many-core architectures. We also note that our approach differs from classical literature proposals oriented to load-sharing or load-balancing in speculative PDES—e.g. Glazer and Tropper (1993), Peluso et al. (2012), Carothers and Fujimoto (2000), Vitali et al. (2012a)—given that they do not consider executions with cross-state access in the definition of dynamic re-balance policies. Our load-sharing proposal allows to significantly reduce the synchronization cost which is often paid when transparently deploying agent-based models supported by PDES engines on multi-core environments. Further, it operates automatically and transparently with respect to the application code.

The new support for load-sharing has been still integrated within the ROOT-Sim open source speculative PDES engine and we present the results of an experimental study showing its benefits for both a synthetic test-case, representative of a class of agent-based models, and *Yades* a real-world demographic simulation framework. The experiments have been run on top of an off-the-shelf machine equipped with 32 CPU-cores.

The remainder of this paper is structured as follows. In Section 2 we discuss our reference PDES architecture. In Section 3 we discuss programming guidelines for ABM with explicit and implicit interactions and we introduce our new load-sharing policy. The experimental assessment of our proposal is presented in Section 4.

2 REFERENCE SPECULATIVE PDES ARCHITECTURE

The reference speculative PDES architecture which we target in our proposal is a Symmetric Multi-Threaded Optimistic Simulation environment, in particular ROOT-Sim, which was introduced in Vitali et al. (2012b) for massively multi-core computing platforms. This is an architecture based on the idea that modern multi-core machines can be seen as *share-everything systems*, in which the most effective way to execute PDES models is to have any worker thread handle whichever concurrent LP. In fact, sharing the data associated with any LP allows the concurrent actors of the PDES system to take care of CPU dispatching pending events in a more inter-twinned fashion. Along this path, other proposals—see, e.g., Santini et al. (2015), Cingolani et al. (2015), Hay and Wilsey (2015), Dickman et al. (2013)—have studied the extreme scenario where the binding between threads and LPs can last no more than the lifetime of an individual event. On the other hand, according to Vitali et al. (2012b), in general application contexts a policy to support *temporary binding* between LPs and worker threads can help fairly distributing the workload, while still exploiting locality. In the most general case, the *rebinding operation*, namely the re-evaluation of the policy to bind LPs to worker threads, is executed either periodically or when triggered by some event internal to the PDES platform (e.g., some threshold on the parameters used by the policy is hit).

As hinted, we also target an innovative programming paradigm for PDES, currently supported by ROOT-Sim, where once a LP is CPU-dispatched for executing some event, the event handler can access the state of any other concurrent LP in place. This paradigm no longer imposes data separation across the LPs, thus enabling so called cross-state dependencies which stand aside of the ones generated via traditional message-exchange, i.e. via cross-scheduling of events. However, correctly handling cross-state dependencies is far from being trivial since it requires that each event is executed as an atomic action involving un-predetermined portions of the overall simulation model state. In Pellegrini and Quaglia (2014b) this is enabled by having a Linux memory management architecture able to capture per-thread accesses to the virtual pages, hosting the state of (possibly) different LPs, and then synchronizing the actions of the threads (at the PDES platform level). These actions allow any cross-state access to operate in isolation on the destination pages, while also observing a correct (although speculatively generated) snapshot of the target LPs' states.

A step ahead in reducing the synchronization overhead has been provided in Marziale et al. (2016), based on the idea to dynamically granulate the different LPs showing large volumes of cross-state interactions along with the different phases of the model execution. These interactions are therefore the fulcrum to establish how to redistribute the workload across the worker threads. In fact, putting multiple LPs which interact via in-place memory accesses on the same thread can reduce the number of memory-page references to be traced. The reason behind this is that the granulated LPs will figure as always coherently advancing in logical time (thanks to LTF scheduling of their events along the thread). Hence, for any thread we only need to trace accesses to pages hosting LPs that are not granulated and bound to the thread itself. This property also reduces the frequency of cross-thread synchronization to be applied for guaranteeing atomicity of their event processing activities whenever a conflicting page access is revealed at runtime. As shown in Marziale et al. (2016), granulation can reduce the synchronization cost, but does not consider that two (or more LPs) operating within different granular LPs can still interact via message-exchange in a

(a) Economic status. (b) Marital status.

Figure 1: Example state transitions for individual-specific explanatory variables.

way that can hamper synchronization (and its overhead), especially when the interactions lead to significant volumes of rollbacks within the speculative execution scheme.

As we will discuss in the next section, the proposed new load-sharing policy takes into account both cross-state and message-exchange interactions, in order to further optimize the fruitful usage of resources, while not imposing any constraint on the original programming model supported by ROOT-Sim. This characteristic will allow making the whole runtime environment much more resilient to performance degradation or reduced scalability due to articulated interactions patterns in the model, as it may be the case for complex and/or large-scale agent-based models coded via the discrete event paradigm. As hinted, providing guidelines on how to fully exploit such a programming model for ABM—in particular for demographic models—is the orthogonal objective of this article.

3 DEMOGRAPHIC AGENT BASED PDES WITH CROSS-STATE AND MESSAGE-EXCHANGE INTERACTIONS

3.1 Guidelines on Simulation Model Coding

In the most general case, the core element of a demographic model is the life course of *individuals*, while their behaviour and their decisions strongly depend on the *environment* they act into Andrew (1998). Therefore, ABM is interesting for demography for its ability to generate personal event histories and producing estimates of the full distribution outcome Montañola-Sales et al. (2014). As a result, two elements need to be presented in any demographic ABM: the environment and the agents (with their interactions). Borrowing from the discussion in Cingolani et al. (2015), in case of an agent-based model developed according to the DES paradigm, we suggest mapping sub-portions of the environment (named *regions*) to LPs, while agents are mapped to specific data structures managed by handlers at regions' LPs—theoretical/abstract agents might be mapped to separate LPs.

As we have already discussed, an agent is likely described in terms of individual-specific explanatory variables. Therefore, changes in the agent state are often expressed as state transitions (implemented within the event handlers' logic) on some variables. An example of a possible state transition for the economic and marital status of a generic agent is depicted in Figure 1. In this specific case, an agent's state could be fully specified by relying on two integer variables. Therefore, the best-suited way to represent an agent is to rely on a data structure (a `struct` in case of C-based programming) which is not directly associated with a specific logic, rather is manipulated by the regions' (LPs') event handlers. This property enables different regions to manipulate the same agent in a different way, thus giving more expressive power at no additional cost. Whenever an agent enters a region (mapped to a specific LP), this can be encoded by having the origin region schedule an event piggybacking the data structure(s) associated with the agent at the destination LP. This LP can therefore register the agents (i.e., the records) within its simulation state. In this case, we note that an LP might implement any logic within its event handlers, and can access any agent currently registered at it.

Anyhow, nothing prevents multiple LPs from keeping in their simulation states the records associated with the same agents. In fact, this reflects a scenario where regions are not necessarily completely disjoint. As an example, one LP might logically represent a city, while another LP might represent a workplace

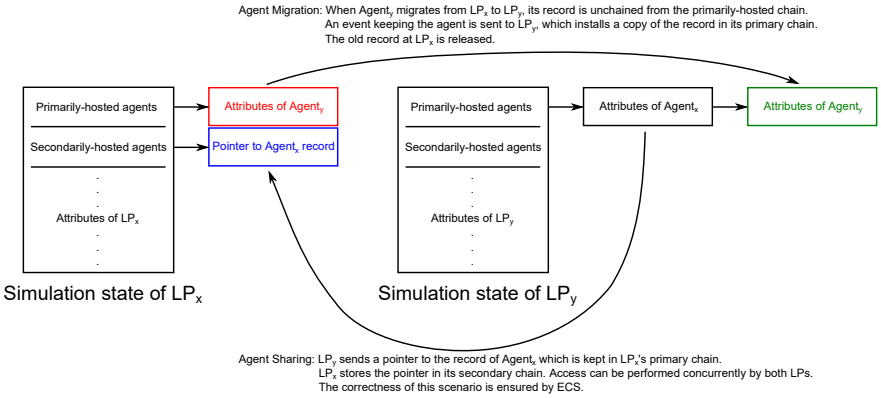


Figure 2: Cross state-enabled programming model for agent-based demographic models.

within it. Both LPs could manage a subset of the overall state transitions which affect an agent, and this organization clearly allows for a simplification of the implementation of the model, and for a higher reuse/interoperability of different simulation models. In this scenario, cross-state synchronization becomes a mandatory aspect to deal with the correctness of the parallel simulation run.

Given the possibility to rely on cross-state synchronization, an overall schematization of this agent-based programming model for demography is depicted in Figure 2. As mentioned, each LP describes either a geographical region or a specific place (e.g., a workplace, a hospital, ...) within one of the geographical regions. Both kinds of LPs keep two lists of records, a *primary list* and a *secondary list*. The primary list is used to keep track of the agents which are currently in the region represented by the LP, and therefore the handlers within the LP can easily manipulate their attributes. Each agent keeps within its record a system-wide unique id, so that the LP can precisely identify a given agent in case an event affects only one or a subset of the agents currently hosted. Similarly, each LP keeps a secondary list. This list keeps track of the agents which can be managed (in terms of record update) by the LP, but they are not *physically* hosted at the region. In particular, this secondary list is a list of *pointers* to some agent records kept in the primary list of any other LP in the system. This allows multiple LPs to share a portion of their simulation state, and concurrently access the records associated with agents which are of interest for the execution of the simulation model. In particular, this allows to decouple different logical aspects of the model. For example, if a LP represents a workplace, all agents working there could have their salary updated via a simple chain traversal—we emphasize that this operation is independent of any other action involving the agents, and is in fact realized on a separate module of the model, concurrently executed at an LP which is separate from the ones hosting the agents.

By this organization of the simulation states of LPs, we envisage two different operations on agents which are of general usability for demographic agent-based models:

- *Agent sharing*: if a LP wants to share an agent with one or multiple regions/entities, it simply sends an event to the corresponding LPs carrying a *pointer* to the record chained to its primary list.
- *Agent migration*: in case an agent physically moves from one region of the model to another, the origin LP creates a copy of the agent's record into a message, which is scheduled to the destination LP with a model-specific timestamp increment. Additionally, the record currently chained to the origin LP's primary list is detached and `free()`'d. Moreover, all the LPs which currently keep a pointer to the record are informed of this migration via message passing (the agent's id is the only information to be communicated) so that the pointers are removed from the secondary list.

If two agents have to interact with each other, this is likely due to them being registered at the same LP (or shared across the same LPs). In this scenario, LPs can easily access the agents' records from the lists. In the more unlikely case that two agents *remotely* interact (i.e., they interact unless they are

not geographically close), this could be anyhow supported via traditional message passing. To this end, the agent (run by its hosting LP) must know in which region the destination agent is currently hosted. Nevertheless, this does not hamper the simplicity of our programming model, as an agent can keep this information within its record. The model should only ensure that when an agent migrates to another region, it informs (via message passing) the interested agents of their migration.

3.2 The Innovative Load-Sharing Policy

The introduced programming model is a general one, and it immediately opens up the possibility to enforce a very effective load-sharing policy. In particular, for each LP_i of the system, we rely on a set of counters, identifying the volume of implicit and explicit interactions towards any other LP_j of the system. In more detail, each LP_i ($i \in [0, maxLP-1]$) is associated with a tuple $\langle I_0, I_1, \ldots, I_{maxLP-1}, E_0, E_1, \ldots E_{maxLP-1} \rangle$ where each component I_j is the amount of implicit accesses from LP_i to LP_j—measured in terms of cross-state synchronizations as measured according to the aforementioned proposal in Pellegrini and Quaglia (2014b). Each E_j is the amount of events scheduled via message passing from LP_i to LP_j. For the case $i = j$, we arbitrarily set the value I_i to the number of events executed by LP_i. This is done under the assumption that whenever LP_i executes an event scheduled towards itself, the likelihood that it will access its own state is very high. This decision prevents the introduction of any bias in the general algorithm which is used for load-sharing.

Each tuple $\langle I_0, I_1, \ldots, I_{maxLP-1}, E_0, E_1, \ldots E_{maxLP-1} \rangle$ can be regarded as a set of coordinates in an n-dimensional space, refered as the *interaction space* among the active LPs in the system. Our load-sharing strategy aims at identifying a set of *clusters of LPs* which show a high inter-dependence among each other. In fact, if two LPs have similar coordinates in the n-dimensional space, they are very likely to interact. The number of clusters to be identified is known a-priori, and corresponds to the number of active worker threads (namely, the amount of CPU cores used on the current machine) which carry on the simulation.

To identify these clusters, we rely on a variant of the Lloyd's solution Lloyd (1982) to the problem of finding evenly-sized Voronoi regions in an Euclidean space. This variant, known as the k-medoids clustering algorithms Kaufman and Rousseeuw (1987), tries to partition the available $maxLP-1$ LPs into K different clusters (where K is the number of CPU cores) trying to minimize the effect of outliers. Specifically, if we call $\mathbf{i}$ and $\mathbf{j}$ the n-dimensional vectors associated with the coordinates of LP_i and LP_j in the n-dimensional interaction space, we define the *distance* between the two LPs as the Manhattan distance $d(\mathbf{i},\mathbf{j}) = \|\mathbf{i}-\mathbf{j}\| = \sum_{i=1}^{n} |i_i - j_i|$. This distance is used in the objective function of the algorithm, which is defined as:

$$D = \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j \in C_k} d_{i,j} \tag{1}$$

where C_k is the set of all LPs in cluster k. When the load-sharing resource allocation is recomputed—this recomputation can be periodical or triggered by a certain event of the system—an initial LP is selected having the shortest distance to any other LP in the n-dimensional space, which is *approximately* in the center. Then, other $k-1$ LPs are selected so that they decrease the value of D as much as possible. In a second phase, possible alternatives for the k objects are selected, by picking an unselected LP and trying to exchange it with one of the k objects. The choice is kept if and only if it produces a decrease in the value of D. This step is repeated until no exchange can be found that lowers the objective function's value. To avoid this procedure take too long, we impose a maximum number of refinement steps, which can be tuned at compile time.

The selected k LPs define the centroids of the k Voronoi regions of the n-dimensional interaction space. The LPs belonging to each group can then be picked minimizing the distance $d(\mathbf{i},\mathbf{j})$ with respect to the centroids.

4 EXPERIMENTAL RESULTS

In this section, we provide a set of experiments to assess the validity of our load-sharing policy for agent-based simulation models with LPs interacting both explicitly and implicitly. As the runtime environment we have used ROOT-Sim, an open source Time Warp-based general purpose simulation platform hosting simulation models developed using the C programming language. The full source code of our implementation is available at https://github.com/HPDCS/ROOT-Sim. This platform already offers the baseline support for cross-state synchronization as presented in Pellegrini and Quaglia (2014b) and the granulation facilities presented in Marziale et al. (2016), and we have extended it by including the proposed load-sharing policy. The hardware architecture used for running the experiments is a 64-bit NUMA machine, namely an HP ProLiant server, equipped with four 2GHz AMD Opteron 6128 processors and 64 GB of RAM. Each processor has 8 cores (for a total of 32 cores) that share a 12MB L3 cache (6 MB per each 4-cores set), and each core has a 512KB private L2 cache. For the parallel runs we configured the simulation platform to use 32 worker threads.

4.1 A Synthetic Demographic Model

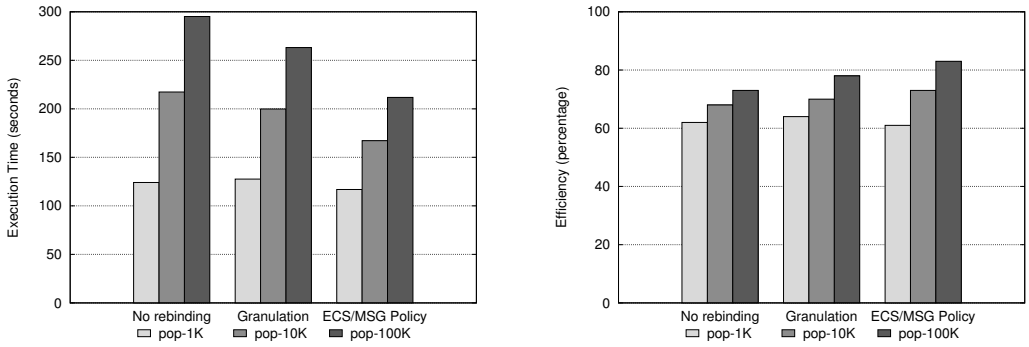
To study to what extent our policy is able to capture complex interactions, we first rely on a synthetic benchmark which is representative of a wide range of agent-based models. Upon simulation startup, a pre-determined number of LPs acting as non-disjoint regions is set up. These LPs implement event handlers which, with a certain probability, operate changes on the hosted agents, execute an agent migration, or schedule to any other LP of the system an *operative event*, namely an event which is associated with an operation that correlates two agents hosted by different cells. This latter event mimics kinship or family interactions, e.g., to relative individuals who live in separate regions of the world.

As described, we map agents to data structures. An agent is described by a bitmask of attributes (which mimic state machines as depicted in Figure 1) and a payload which is updated by the event handler implemented at any LP. In particular, we define three operations:

- *State-machine update*: with a certain probability p_{smu}, a bit in the bitmask is negated, mimicking a transition in the state of the agent;
- *Memory update*: with a certain probability p_{mu}, a portion of the payload of the agent's structure is written with random data, mimicking the update of less-concise metadata describing the agent;
- *Remote agent interaction*: with a certain probability p_{rai}, a random LP is selected, and an event piggybacking some random data is scheduled to it. Upon the receipt of this event, a random agent is picked and the content of the message is copied into its state. This operation mimics complex interactions which can affect agents in remote portions of the simulated environment.

Upon simulation startup, each LP instantiates a set of agents, so that they are initially evenly distributed across regions. Agents are organized into two chains: a *primary chain* keeps pointers to all the agents which have been directed (or instantiated) towards the current LP. A *secondary chain* keeps pointers to agents which reside in the given LP as the result of a sharing operation. Agents from the secondary chain cannot be directly migrated by the LP, rather they are only removed whenever the agent migrates from the primary region—this is supported by having the primary region notify this migration via message passing.

Each LP schedules to itself separate chains of events, exponentially distributed, which trigger the state-machine update, and the memory update operations. Once one of these operations is triggered, the LP scans the whole list of records so as to randomly select agents which undergo the corresponding operation. After a certain residence time, an agent is migrated towards one remote region. When this agent is registered, a new agent migration event is scheduled, so that its lifetime within a certain region is pre-determined. When this lifetime expires, the agent is migrated towards another region. Upon installation, with a certain probability p_{sh} the agent is shared (via message passing) with another region as well.



(a) Execution time (b) Efficiency

Figure 3: Synthetic demographic model results.

In Figure 3, we report performance data from our synthetic experiments. We set $p_{smu} = 0.3$, $p_{mu} = 0.5$, and $p_{rai} = 0.2$, while the number of regions is set to 512. We varied the population size from 1,000 agents to 100,000 agents, and we measured the total execution time and the simulation's efficiency (percentage of productive work of speculative runs) in three different setting: using no rebinding policy, employing the granulation facility offered by Marziale et al. (2016), and making use of our innovative load-sharing policy. For both basic granulation and the new load-sharing policy the rebinding of LPs to worker threads is carried out each 10 wall clock time seconds. From the results we can see that an increase in the population size shows an increase in the efficiency of the simulation. This is related to the fact that the average granularity of events increases as well—it moves from 10 μs up to 45 μs. In fact, since we do not vary the number of regions, the average number of agents registered at each region is likely to increase. Anyhow, in case the execution is supported by our innovative load-sharing policy, this effect is more evident. This is reflected into a scaled up gain, in terms of execution speed, when running larger models with such a policy. From the data we can infer that the advantage from the proposed load-sharing policy over the original LP granulation scheme is of up to 20%. Furthermore, the results show we increase the performance by up to 40% with respect to the settings where no workload rebinding action across the worker threads is executed by the PDES environment.

4.2 Yades

Yades (*Yet Another Demographic Simulator*) Onggo (2008), Onggo et al. (2010) is a parallel demographic agent-based simulation tool. It uses an agent-based approach to model the life course of individuals in an environment formed by a set of regions through five demographic components: fertility, which determines whether a female individual will give birth based on her current characteristics and the calendar time; mortality, understood as the time when an individual will die or his survival; change in economic status by using a state diagram with the ability to include explanatory variables such as age, family composition or socio-economic factors, marital status, also based on the characteristics of the individual(s) and the current calendar time by using state-transition diagrams, and a match-making function to form new pairs, and migration (either domestically or internationally) which can combine individual, family or regional specific factors to explain the individual's decision to migrate to a new place. In Yades, a rich set of attributes is used coming from census, surveys. Moreover, qualitative data can be included in the model through behavioural rules that help to overcome some data-related limitations of over-reliance on purely statistical information.

In Figure 4 we report the speedup with respect to the sequential execution when simulating the demographic evolution of Gambian immigrants in Spain during 10 years Montañola-Sales et al. (2014), considering 40,000 families and a migration probability set to 10%. The results show that supporting the execution of this simulation with our innovative load-sharing policy provides a non-negligible performance

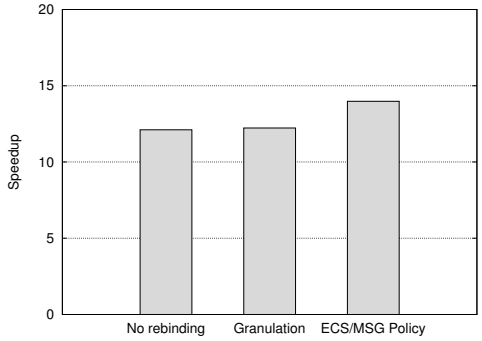


Figure 4: Speedup results with Yades.

increase, considering that no modification at all has been made on the simulation model. At the same time, the performance increase offered by the granulation support alone has limited impact. The reason behind this is that Yades still relies much on traditional message-passing facilities, which have been the de-facto standard for PDES in the past years.

5 CONCLUSIONS

In this article we have investigated how recent programming models for PDES, no longer limited to data-separated accesses across concurrent LPs, can be exploited for demographic agent-based models. We have discussed model coding aspects, when cross-LP in-place state accesses are enabled by the underlying PDES engine. Also, we have presented a load-sharing policy for the case of speculative PDES execution with both cross-state and message-exchange interactions, which has revealed effective with both synthetic and real-world demographic agent based simulations. The policy stands anyhow as a general purpose one, and we plan to test it with applications from different domains as future work.

REFERENCES

Andrew, H. 1998. *Demographic Methods*. Routledge.

Balbo, A. L., X. Rubio-Campillo, B. Rondelli, M. Ramírex, C. Lancelotti, A. Torrano, M. Salpeteur, N. Lipovetzky, V. Reyes-García, C. Montañola-Sales, and M. Madella. 2014. "Agent-based simulation of Holocene monsoon precipitation patterns and hunter-gatherer population dynamics in semi-arid environments". *Journal of Archaeological Method and Theory* 21 (2): 426–446.

Carothers, C. D., and R. M. Fujimoto. 2000. "Efficient execution of Time Warp programs on heterogeneous, NOW platforms". *IEEE Transactions on Parallel and Distributed Systems* 11 (3): 299–317.

Cingolani, D., A. Pellegrini, and F. Quaglia. 2015. "RAMSES: Reversibility-based agent modeling and simulation environment with speculation support". In *Proceedings of Euro-Par 2015: Parallel Processing Workshops*, edited by S. Hunold, A. Costan, D. Ginenéz, A. Iosup, L. Ricci, M. E. Gómez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, PADABS, 466–478. LNCS, Springer-Verlag.

Dickman, T., S. Gupta, and P. A. Wilsey. 2013. "Event pool structures for PDES on many-core Beowulf clusters". In *Proceedings of the 2013 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, 103–114: ACM Press.

Fujimoto, R. M. 1990. "Parallel discrete event simulation". *Communications of the ACM* 33 (10): 30–53.

Glazer, D. W., and C. Tropper. 1993. "On process migration and load balancing in Time Warp". *IEEE Transactions on Parallel and Distributed Systems* 4 (3): 318–327.

Hay, J., and P. A. Wilsey. 2015. "Experiments with hardware-based transactional memory in parallel simulation". In *Proceedings of the 2015 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 75–86. New York, New York, USA: ACM Press.

Jefferson, D. R. 1985. "Virtual Time". *ACM Transactions on Programming Languages and System* 7 (3): 404–425.

Jennings, N. R., K. Sycara, and M. Wooldridge. 1998. "A roadmap of agent research and development". *Autonomous agents and multi-agent systems* 1 (1): 7–38.

Kaufman, L., and P. J. Rousseeuw. 1987. "Clustering by means of medoids". *Statistical Data Analysis Based on the L1-Norm and Related Methods*:405–416416.

Lloyd, S. 1982, mar. "Least squares quantization in PCM". *IEEE Transactions on Information Theory* 28 (2): 129–137.

Luke, S., C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. 2005. "MASON: A multiagent simulation environment". *Simulation* 81 (7): 517–527.

Lysenko, M., and R. M. D'Souza. 2008. "A framework for megascale agent based model simulations on the GPU". *Journal of Artificial Societies and Social Simulation* 11 (4): 10.

Marziale, N., F. Nobilia, A. Pellegrini, and F. Quaglia. 2016. "Granular Time Warp objects". In *Proceedings of the 2016 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 57–68. New York, New York, USA: ACM Press.

Minar, N., R. Burkhart, C. Langton, and M. Askenazi. 1996. "The SWARM simulation system: A toolkit for building multi-agent simulations". Technical report, Santa Fe Institute.

Montañola-Sales, C., J. Casanovas-Garcia, A. Kaplan-Marcusán, and J. M. Cela-Espín. 2014. "Demographic agent-based simulation of Gambians immigrants in Spain". In *Proceedings of the 10th Social Simulation Conference*: European Social Simulation Association.

North, M. J., T. R. Howe, N. T. Collier, J. R. Vos, and J. V. M.J. North, T.R. Howe, N.T. Collier. 2005. "The Repast simphony runtime system". In *Proceedings of the Agent 2005 Conference on Generative Social Processes, Models and Mechanisms*, 151–158: Argonne National Laboratory.

Onggo, B. S. S. 2008, dec. "Parallel discrete-event simulation of population dynamics". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, 1047–1054. IEEE Computer Society.

Onggo, B. S. S., C. Montañola-Sales, and J. Casanovas-Garcia. 2010. "Performance analysis of parallel demographic simulation". In *Proceedings of the 24th European Simulation and Modelling Conference*, ESM, pp. 142–148: Eurosis-ETI.

Page, S. E. 2008. "Agent-based models". In *The New Palgrave Dictionary of Economics*, edited by S. N. Durlauf and L. E. Blume, 47–52. Nature Publishing Group.

Park, H., and J. Han. 2008. "Fast rendering of large crowds using GPU". In *Entertainment Computing*, 197–202. Springer Berlin Heidelberg.

Pellegrini, A., and F. Quaglia. 2014a. "The ROme OpTimistic Simulator: A tutorial". In *Proceedings of the Euro-Par 2013: Parallel Processing Workshops*, edited by D. an Mey, M. Alexander, P. Bientinesi, M. Cannataro, C. Clauss, A. Constan, G. Kecskemeti, C. Morin, L. Ricci, J. Sahuquillo, M. Schulz, V. Scarano, S. L. Scott, and J. Weidendorfer, PADABS, 501–512. LNCS, Springer-Verlag.

Pellegrini, A., and F. Quaglia. 2014b. "Transparent multi-core speculative parallelization of DES models with event and cross-state dependencies". In *Proceedings of the 2014 ACM/SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS, 105–116: ACM Press.

Peluso, S., D. Didona, and F. Quaglia. 2012, nov. "Supports for transparent object-migration in PDES systems". *Journal of Simulation* 6 (4): 279–293.

Prats, C., C. Montañola-Sales, J. F. Gilabert-Navarro, J. Valls, J. Casanovas-Garcia, C. Vilaplana, P.-J. Cardona, and D. López. 2016, jan. "Individual-based modeling of tuberculosis in a user-friendly interface: Understanding the epidemiological role of population heterogeneity in a city". *Frontiers in Microbiology* 6 (1564).

Santini, E., M. Ianni, A. Pellegrini, and F. Quaglia. 2015, dec. "Hardware-transactional-memory based speculative parallel discrete event simulation of very fine grain models". In *2015 IEEE 22nd International Conference on High Performance Computing*, HiPC, 145–154: IEEE.

Takahashi, T., S. Tadokoro, M. Ohta, and N. Ito. 2002. "Agent based approach in disaster rescue simulation-From test-bed of multiagent system to practical application". In *Robot Soccer World Cup V*, RoboCup, 102–111: Springer-Verlag.

Tisue, S., and U. Wilensky. 2004. "Netlogo: A simple environment for modeling complexity". In *Proceedings of the International Conference on Complex Systems*, ICCS, 1–10: NECSI.

Vitali, R., A. Pellegrini, and F. Quaglia. 2012a, jan. "Load sharing for optimistic parallel simulations on multi core machines". *ACM SIGMETRICS Performance Evaluation Review* 40 (3): 2–11.

Vitali, R., A. Pellegrini, and F. Quaglia. 2012b, jul. "Towards symmetric multi-threaded optimistic simulation kernels". In *Proceedings of the 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS, 211–220: IEEE Computer Society.

AUTHOR BIOGRAPHIES

ALESSANDRO PELLEGRINI is a Research Fellow at Department of Computer, Control, and Management Engineering at Sapienza, University of Rome, where his main research topics are high-performance simulation system, cloud computing, compiling and executable manipulation techniques, non-blocking algorithms, and speculative synchronization techniques for high-performance computing. He is an active member in the international research community, serving as a referee for several international conference and journals, and as a TPC member for prestigious international conferences. His email address is pellegrini@dis.uniroma1.it.

CRISTINA MONTAÑOLA-SALES is a researcher at inLab FIB in the Barcelona School of Informatics (FIB). She also is adjunct associate professor at the Department of Statistics and Operations Research at the Universitat Politcnica de Catalunya (UPC). She also carries out her research at the Barcelona Supercomputing Center (BSC). She holds a PhD in Statistics and Operations Research and BSc, MSc in Computer Science from the UPC. Her research interests include agent-based modeling, computer simulation, high-performance computing and computational social science. Her email address is cristina.montanola@upc.edu.

FRANCESCO QUAGLIA is an Associate Professor at the School of Engineering of the University of Rome "La Sapienza". His research interests span from theoretical to practical aspects concerning distributed systems and applications, distributed protocols, middleware platforms, parallel discrete event simulation, federated simulation systems, parallel computing applications, fault-tolerant programming, transactional systems, operating systems, web-based systems and performance evaluation of software/hardware systems. In these areas he has published more than 160 technical articles. His email address is quaglia@dis.uniroma1.it.

JOSEP CASANOVAS-GARCÍA is a Full Professor in Operations Research, specializing in Simulation Systems. He is one of the founders of the Barcelona School of Informatics (FIB). He is also the director of inLab FIB, a research lab particularly active in innovation and technology transfer to business. He has led several projects in the area of simulation. Currently, Josep Casanovas is coordinator of the Severo Ochoa Research Excellence Program in the Barcelona Supercomputing Center (BSC-CNS). His email address is josepk@fib.upc.edu.