PDES-MAS : distributed simulation of multi-agent systems

Suryanarayanan, Vinoth; Theodoropoulos, Georgios; Lees, Michael

2013

Suryanarayanan, V., Theodoropoulos, G., & Lees, M. (2013). PDES-MAS : distributed simulation of multi-agent systems. Procedia computer science, 18, 671-681.

https://hdl.handle.net/10356/98474

https://doi.org/10.1016/j.procs.2013.05.231

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 671 – 681

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

PDES-MAS: Distributed simulation of multi-agent systems

Vinoth Suryanarayanan[a], Georgios Theodoropoulos[b,*], Michael Lees[c]

[a]School of Computer Science, University of Birmingham,UK
[b]Institute of Advanced Research Computing, Durham University,UK
[c]School of Computer Engineering, Nanyang Technological University,Singapore

Abstract

Multi-agent systems (MAS) are increasingly being acknowledged as a modelling paradigm for capturing the dynamics of complex systems in a wide range of domains, from system biology to adaptive socio-technical system of systems. The execution of such MAS simulations on parallel machines is a challenging problem due to their dynamic, non-deterministic, data-centric behaviour and nature. These problems are exacerbated as the scale of such MAS models increases. PDES-MAS is a distributed simulation kernel developed specifically to support MAS models addressing the problems of partitioning, load balancing and interest management in an integrated, transparent and adaptive manner. This paper presents an overview of PDES-MAS and for the first time it provides a quantitative evaluation of the system.

Keywords: agent-based systems; distributed simulation.

1. Introduction

The last decade has witnessed an explosion of interest in complex systems, which involve dynamic and unpredictable interactions between large numbers of components including software, hardware devices (such as sensors), and social entities (people or collective bodies). Examples of such systems range from traditional embedded systems, to systems controlling critical infrastructures, such as defence, energy, health, transport and telecommunications, to biological systems, to business applications with decision-making capabilities, to social systems and services, such as e-government, e-learning, etc. The complexity of such systems renders simulation modelling the only viable method to study their properties and analyse their emergent behaviour. Multi-agent systems (MAS) are acknowledged as a particularly suitable paradigm for modelling complex systems.

As MAS models are being applied for the analysis of ever larger complex adaptive System-of-Systems, distributed simulation is emerging as the only viable approach to deal with their size and scale. However conventional distributed simulation approaches are not well suited for MAS simulations. The fundamental problem is that MAS models are by nature data-centric. A MAS simulation typically requires a very large set of shared variables which could, in principle, be accessed or updated by the agents (if they were in the right position at the right time etc.) [1]. Encapsulating the shared state in a single process (i.e., via some centralised scheme) introduces a bottleneck, while distributing it all across the processes of the simulation (decentralised, event driven scheme) will typically

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

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.231

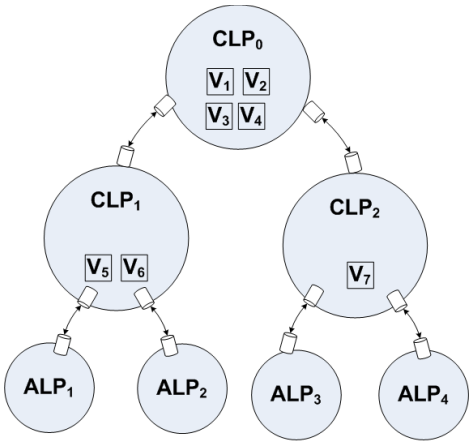


Fig. 1: A tree of 3 CLPs and 4 ALPs.

result in frequent all-to-all communication and broadcasting. Over the last few years, several approaches to distributed simulation of MAS models have been proposed,a survey of the most influential of them are reported in [2]. Relevant research has also been undertaken in the context of Data Distribution Management for large scale distributed simulation (HLA [3, 4]) and Distributed Virtual Environments. A pioneering effort in this direction has been PDES-MAS, a distributed simulation framework specifically designed to support large scale MAS models [1]. PDES-MAS provides mechanisms for the adaptive partitioning of the simulation shared state and addresses the problems of load balancing, synchronisation and interest management (avoiding all to all communications) in an integrated, adaptive and fully transparent manner. This paper provides a description of the PDES-MAS system and for the first time presents a quantitative analysis of the performance of the system.

The rest of the paper is organised as follows: Section 2 provides an overview of the philosophy of PDES-MAS while section 3 gives an insight of the internal architecture of the system. Section 4 provides a quantitative analysis of the performance of the system in two parts: first a holistic analysis of different aspects of the system under different workload conditions and then a scalability analysis. The paper concludes in section 5.

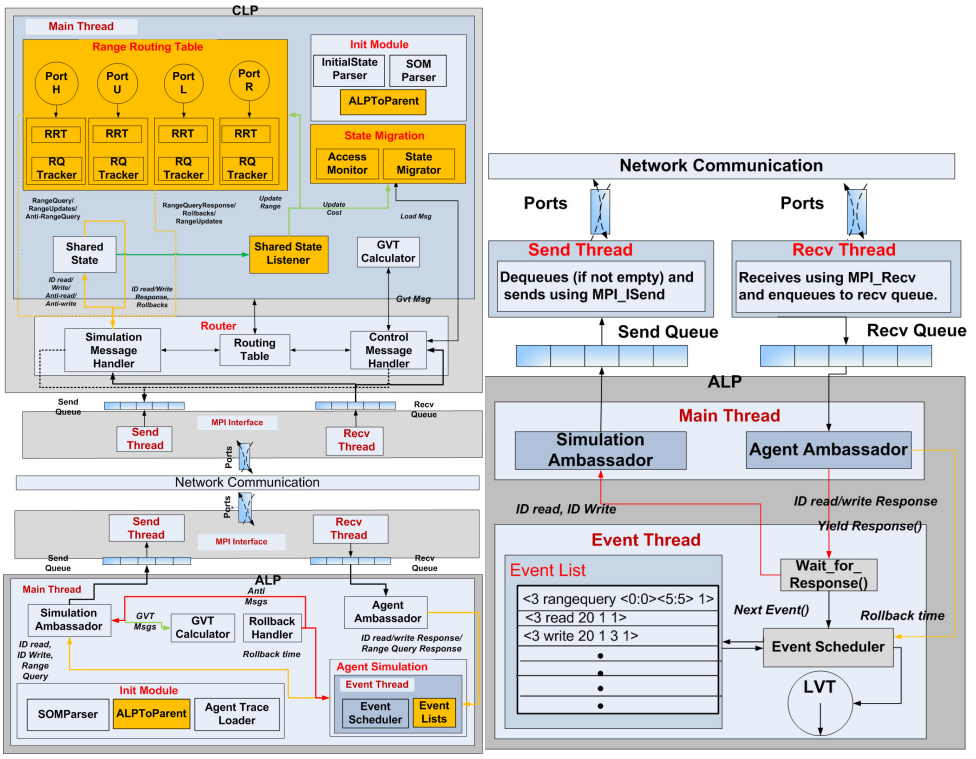
2. PDES-MAS

The PDES-MAS framework is based on PDES paradigm, where a simulation model is divided into a network of concurrently executing Logical Processes (LPs), each maintaining and processing a disjoint state spaces of the model. Two types of LP exist in a PDES-MAS simulation. *Agent Logical Processes* (*ALPs*) are responsible for modeling the behaviour of the agents in the MAS. This includes the processing of sense data, the modeling of behavioural processes (such as planning or ruleset evaluation) and the generation of the new actions. ALPs store only private state variables while the shared state (public variables, including publicly accessible attributes of agents) are distributed over and managed by a a tree-like network of server LPs known as *Communication Logical Processes* (*CLPs*) as depicted in figure 1. CLPs essentially implement a space-time Distributed Shared Memory (DSM) model whereby agents interact with their environment and communicate via accessing public variables.

In response to the sensing and acting of agents in the simulation, ALPs perform reads and writes on SSVs in the system. ALP reads come in two forms: ID-Queries and Range-Queries (sense). The former read the value of a particular state variable while Range-Queries get the IDs of all SSVs whose value satisfies a given predicate.

A CLP is responsible for synchronising the events it receives from ALPs. To facilitate optimistic synchronisation, public variables are implemented as *Shared-State Variable* (*SSV*) data-structures which store the history of values taken by a particular variable over time [5].

The primary philosophy of PDES-MAS is to provide multiple ALPs concurrent access to a set of SSVs in a scalable manner by a balanced distribution of SSVs. Accesses to SSVs held by remote CLPs are forwarded until they reach their destination. Communication between the LPs in the system is realised by *ports*, which are complex



(a) PDES-MAS architecture (b) Event flow inside an ALP.

Fig. 2: PDES-MAS Internal Architectural Elements

data-structures maintaining routing information. The CLP tree is reconfigured dynamically and automatically to reflect the interaction patterns between the agents and their environment so that SSVs which are accessed most frequently by a given ALP are as close as possible to that ALP in the tree. The aim is to concurrently minimise the overall cost for accessing SSVs in the system. The following section provides an outline of the internal architecture of PDES-MAS processes.

3. Architecture of PDES-MAS

PDES-MAS is implemented using the C/C++ programming language and is compatible with both *LAM/MPI (7.1.1)* and *openMPI-1.4.3* libraries. The internal architecture of the PDES-MAS processes is presented in figure 2a. As mentioned in the previous section, the PDES-MAS CLP-tree is agnostic of the internal implementation of the Agent processes (ALP). In [6] we have presented the architecture of an ALP process to support the interactive simulation of medieval military logistic models as part of the MWGrid project[1]. The ALP in figure 2a has been designed to support trace driven simulations that have been utilised for the experimental analysis presented in this paper.

3.1. Initialisation

PDES-MAS bootstraps loading with the knowledge of the initial state of the simulation. This includes:

1. A list of SSV IDs, their data types and initial values.
2. A list of simulation event types. For example, an agent moving an object in the simulation is modeled as a *WRITE MESSAGE* with a new value to the object.

[1] `http://www.ahessc.ac.uk/manzikert`

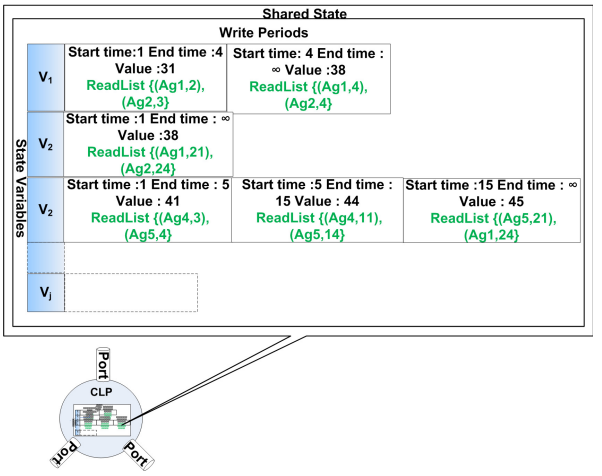


Fig. 3: Maintaining Shared State Variables (SSVs) in a CLP

3. A list of ALP IDs and their locations in the CLP tree.

The **InitialStateParser** module loads with a list of SSVs IDs and their initial values. An SSV ID is associated with a unique format *(ObjectInstanceId, VariableClassId)*. *ObjectInstanceId* represents a type of an object in the simulation (eg. location) and *VariableClassId* represents an attribute type of the object (eg. a (x,y) 2D position). This information is stored in a XML file, which is then parsed and loaded into a CLP. The **SOM Parser (Simulation Object Model)** module, a concept adopted by HLA technology [3, 4], prescribes the types of simulation objects, attributes and interactions in the simulation model. The interactions (messages) in this framework are of two types: *Simulation Messages* (namely: ID Read, Range Query, ID Write, Rollback, Anti-Messages (read/write) and Response messages) and *Control Messages* (related to GVT calculation and Load management). Similarly to CLPs, the SOM parser module in an ALP loads simulation objects, attribute types and interaction types. Each ALP loads a set of events (actions in the simulation) using *Agent Trace Loader* module. An ALP can hold action lists of several agents. Any agent simulation model which follows the *sense-think-act* cycle can fit in this framework as long as it adheres to the following action message (trace in this particular implementation) formats:

- ID Read:<agent_ id, read, object_ instance_ id, variable_ class_ id, timestamp >,where object_instance_id and variable_class_id define the SSV id.
- ID write: < agent_id, write, object_instance_id, variable_class_id, value, timestamp>
- Range Query:< agent_id, rangequery, minimum_ range, maximum_ range, timestamp >

The events are generated concurrently from ALPs and are forwarded to the CLP tree. The framework supports access to variables of the following data types INTEGER, FLOAT, STRING, LONG, DOUBLE and 2-D POINT of integer type. To provide an indication of the relative time spent at the initialisation phase, in the experimental setup used in this paper, it took on average 0.3 msec to initialise an SSV in a CLP and 0.05 msec to load a trace in an ALP.

3.2. Message Handling

Each CLP has three possible ports *U*, *R* and *L* ('up', 'left' and 'right') connected to neighbouring LPs as well as a local port *H*('here'). Within a CLP, 5 different queues handle messages being sent and received through different ports. Two queues (*Send and Receive Load*) are dedicated to send and receive Load Messages with higher priority, a *Send Control Queue* to send control messages and two normal queues (*Send and Receive*) to send and receive simulation messages with the least priority.

The framework uses MPI-2 libraries to establish communication between LPs. The *MPI Interface* module implements two threads, namely a *Send* thread and a *Receive* thread, to send and receive messages from other

LPs. Two threads work asynchronously to enqueue and dequeue messages from the queues (based on the priority mentioned above) which are then processed within a CLP using a *Main thread*. Semaphores are used to synchronise between *Main thread* and *MPI Interface* threads. Indicatively, if an ID Read for a SSV arrives at a port, the *MPI Interface* queues the message to the Receive (normal) queue and signals Main thread to process the message. The message is then forwarded to the destination port. If the SSV is maintained locally (port - H), then the *Shared State* module (a list of SSVs with its write periods, see section 3) processes the request and constructs a response message, which is buffered into the Send (normal) queue, and signals *MPI Interface* module to send the message. The Main thread will not dequeue the next message until the existing message is processed and queued to the send queue. This synchronisation mechanism ensures that the *Shared State* module handles only one request at any time. The event flow inside an ALP is depicted in figure 2b. The communication of an ALP with the CLP tree is established via two structures:

1. *Simulation Ambassador* transforms an event from an ALP into a message format and forwards to its parent CLP. Since *SOM parser* is pre-loaded with the interactions to the simulation model, the transformation is quite straightforward.
2. *Agent Ambassador* takes care of handing over the response messages to appropriate modules.

Within an ALP, an *Agent Simulation* module models the life cycle of an agent. It maintains event lists and an event scheduler to update the LVT (Local Virtual Time) of an agent. An *event thread* runs separately to maintain and schedule events from the event lists. Events from multiple agents are processed synchronously in a round-robin fashion and their LVTs are maintained separately. An *event thread* dequeues an event of an agent, updates its LVT, converts it to a message using *Simulation Ambassador* and enqueues it to the send queue. The event scheduler waits for response (in case of ID read, write/range query). When an ALP receives a response message, the *Main thread* signals the waiting event scheduler to yield and process the next event. The scheduler does not wait for other events such as GVT. Only a rollback message can interrupt this cycle and resets the scheduler of the rollback agent to the rollback time (see section 3.3).

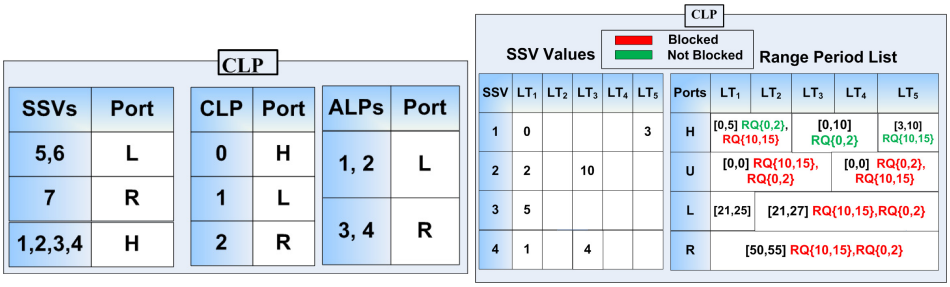
3.3. Synchronisation

PDES-MAS employs optimistic synchronisation to synchronise the events on a SSV. Synchronisation algorithms that have been developed for PDES-MAS are reported in [7, 8, 9, 5]. Rollbacks are triggered when events are processed out of order on a SSV. Typically, a rollback scenario would be an arrival of straggler write arriving at time t′ on a SSV invalidating a ID-Read or Range-Query issued for that SSV at time $> t'$. To enable rollbacks, each SSV is associated with a list of *Write Periods* representing the values taken by the variable at different logical times through the simulation, as illustrated in figure 3. If a write period is subsequently invalidated by a straggler write, any ALPs which read that period must by rolled back. However, invalidating a Range Query requires both time and value of a SSV to ascertain a rollback scenario. To enable such mechanism, the framework maintains a history of actions on a SSV (ID Read/Write) or list of SSVs (Range Queries) in the Write Periods and Range Periods modules. A comprehensive description of the mechanisms employed by PDES-MAS to handle Range Queries is provided in [10]. In PDES-MAS, rollbacks are triggered on three occasions:

- Straggler Writes invalidating ID Reads or Range Queries,
- Straggler Range Updates arriving from external ports invalidating Range Queries ([10]) and
- Initiation of State Migration algorithm to synchronise Range Periods across the CLP tree (see section 3.5).

In these cases, a rollback message is generated to agents that issued premature reads. The *Rollback Handler* within an ALP, on receiving a rollback message for an agent (e.g. at time t′), checks its validity (if $t' < LVT$ of the agent, otherwise rollback ignored). If the rollback is valid, the handler holds the event scheduler from processing any event, sends anti-messages for all events with time $> t'$ and updates the scheduler with the rollback time t' for that agent. The rollbacks are handled for each individual agents.

As is typical with optimistic synchronisation, garbage collection is utilised to free memory used for keeping historical state information. Several algorithms have been proposed to get a snapshot of the system and determine the removable states [11, 12, 13, 14]. PDES-MAS implements an adaptation of Mattern's GVT algorithm [12].



(a) Routing table of CLP id 0.

(b) An example of a Range Period List (at the right) of CLP 0 at ports H, L, U, R evolved over time.

Fig. 4: Routing information in PDES-MAS

3.4. Routing

The system should be capable of routing access queries from ALPs through the tree to the CLP that hosts and maintains the corresponding SSVs. As already mentioned, a message arriving at a port in a CLP will be either processed locally (port - *H*) or forwarded through any of the external ports (*U*, *R*, *L*). Different routing strategies for PDES-MAS have been investigated in [15]. The current implementation of PDES-MAS is utilising the range-based approach whose main idea is that each CLP port records the complete value range of all SSVs that can be found beyond it as well as the port though which a particular SSV can be reached.

Routing is performed by the ports. Entries of the routing tables, which are dynamically updated, have the form of: *(CLP id, Port)*, *(SSV id, port)* and *(ALP id, port)*. Each ALP is identified with the location of its *ParentCLP* (the CLP to which an ALP is attached), this information is maintained by the *AlpToParent* module. Figure 4 presents routing tables of CLP id 0 within the tree configuration presented in figure 1.

To support Range-Queries, CLP maintain range information (min,max) of values of SSVs that can be reached through each port. To enable rollbacks, historical information is maintained about the validity of a range of values of SSVs over a logical time period (Range Periods). More detailed information about the mechanism implemented in PDES-MAS to support Range Queries is provided in [10].

3.5. State Migration

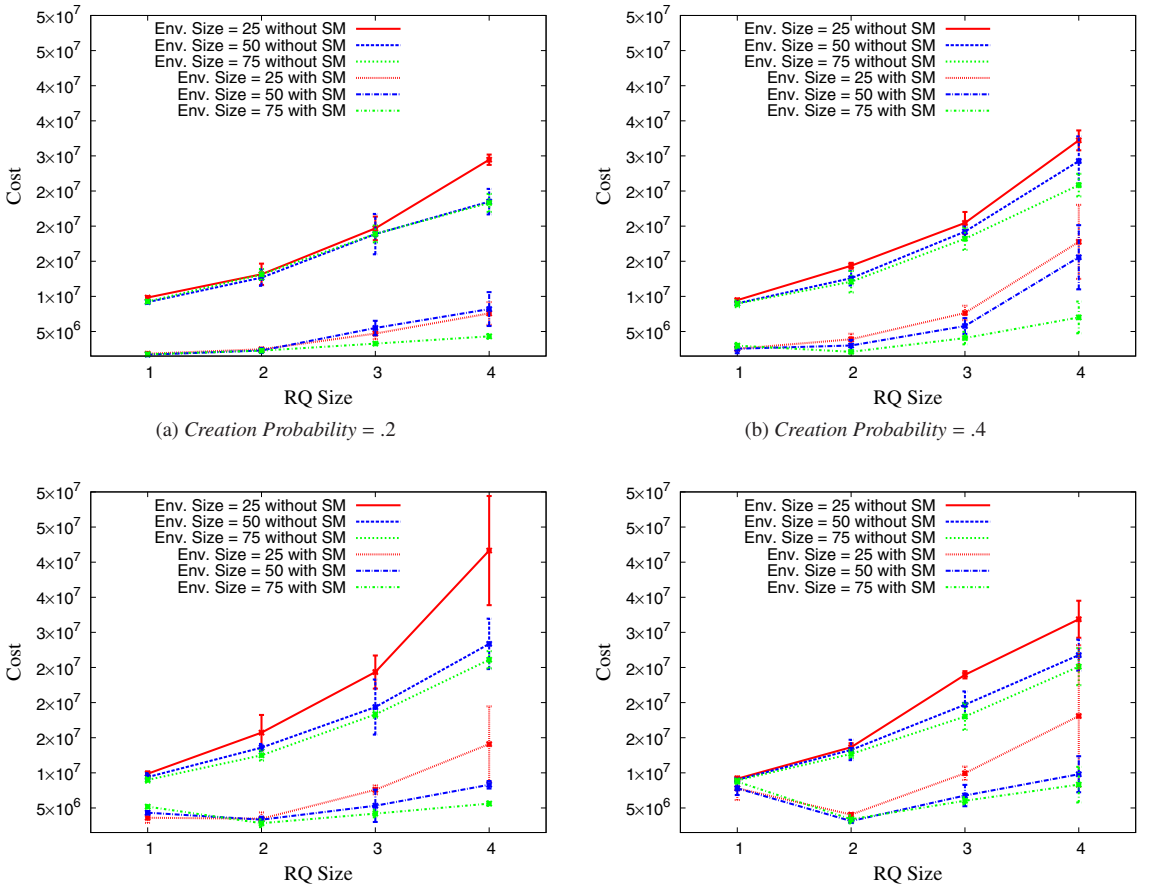
As mentioned in section 2, in the current implementation of PDES-MAS, the CLPs are configured in a fixed tree-structure of a pre-determined size while the SSVs migrate through the tree closer to the agents (ALPs) that access them. The aim is to reduce the total access cost of the simulation, defined as the sum of the access cost of SSVs, where access cost of a SSV is a product of number of accesses with hops required to access it. SSV migration is achieved by means of a competitive optimisation algorithm reported at [16]. Based on this algorithm, a CLP monitors the access patterns at SSVs it maintains through each port and decides whether an SSV should stay in the CLP or should be migrated through the port it is accessed more often. To avoid thrashing, different thresholds are used for SSV migration. The *Access Monitor* module monitors access patterns through different ports on SSVs using *SharedStateListener* module. The concept is similar to listener design pattern, where state changes in shared state of a CLP (ID read/Write/Range-Query) is automatically informed to all registered listeners using the *SharedStateListener* module. The *shared state* module updates all listeners with an access type (ID Read/Write/Range-Query), arriving port and number of hops (measurement is taken from the message itself) [17].

Initiation of a State Migration (SM) generates extra rollbacks to synchronise Range Periods across the CLP tree. An analysis of the impact of such rollbacks is quantified and presented in [17]. The advantage of this mechanism is that migration procedure does not affect any transient messages in the simulation.

4. Experimental Analysis

This section presents an evaluation of the PDES-MAS system under different conditions and CLP-tree configurations. As a benchmark, we have used Tileworld [18], an established and widely accepted agent-based testbed

used extensively to study agent's reasoning and committing strategies. In Tileworld, agents move in a 2-D grid environment of squared cells which contains objects such as tiles, holes and obstacles. The objective of an agent is to maximise its score by pushing the tiles into the holes while avoiding obstacles. Objects in the environment appear and disappear dynamically based on user controlled parameters. In each step, an agent goes through a 'sense-think-act' cycle, prioritising the selected targets within its sensor range, and then moving to its target using A* route planning with a minimum cost (defined with the number of cells required to reach a target). An agent with sensory range r can cover **8(1+2+...+r) cells**, but it can only move in four directions *(UP, LEFT, RIGHT, DOWN)*.



(a) *Creation Probability* = .2 (b) *Creation Probability* = .4

(c) *Creation Probability* = .6 (d) *Creation Probability* = .8

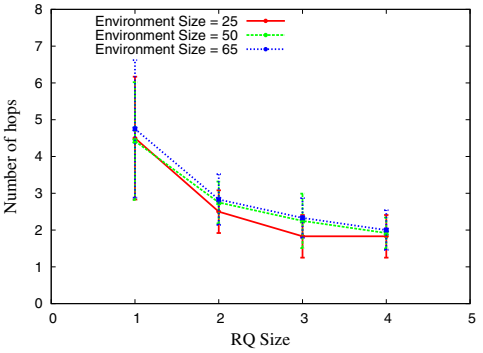
Fig. 5: Access Cost for varying *RQ size* and *Creation Probability* values with their standard deviations.

The computational infrastructure used for the experiments was the BlueBEAR cluster environment[2]. Each worker node of teh cluster is a dual-processor dual-core (4 cores/node) 64-bit 2.6 GHz AMD Opteron comprising 8GB of memory. All worker nodes ran Scientific Linux version 5.2 and communicate via an Infiniband interconnect. For the experiments *openMPI (1.4.3)*[19] libraries were utilised.

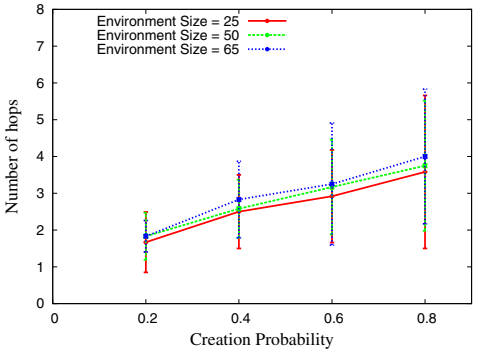
4.1. Holistic Analysis

Our analysis is focussing on the influence of three simulation parameters such as *RQ size*, *Creation Probability* and *Environment Size* in the system measured by three metrics: *Access Cost, Range Query Propagation and*

[2] `http://www.bear.bham.ac.uk/bluebear/`



(a) Average Update propagation for varying *RQ size* and *Environment Size* values with their standard deviations (Error bars) over *Creation Probability* values



(b) Average Update propagation for varying *Creation Probability* and *Environment Size* values with their standard deviations (Error bars) over *RQ size* values

Fig. 6: Average Update propagation

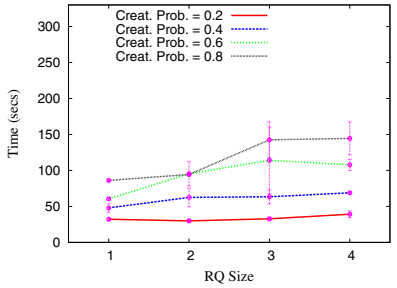
Simulation Time. For our analysis, the PDES-MAS system was configured as a CLP-tree of 7 CLPs. The objects (Tiles, holes, rocks and agents themselves) are modeled as SSVs. All SSVs are initially placed at the root CLP to more clearly show the effect State Migration has on performance. 16 ALPs, four attached to each leaf CLP. Simulations are run for 300 logical time ticks (or steps). There are 10 random seeds used for each simulation parameter.

4.1.1. Access Cost and Query/Update Propagation

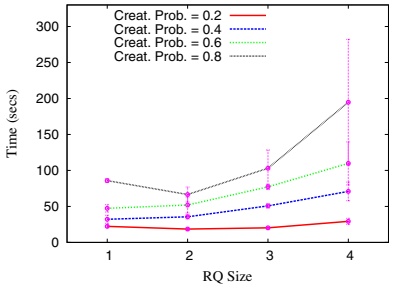
The impact of accessing SSVs under different conditions is measured by two metrics: Access Cost and Query/Update propagation. The Access Cost of an SSV is the product of the number of accesses by the number of hops required for each access to fetch that SSV. The total access cost of the simulation is measured by the sum of the access cost over all SSVs. Figure 5 presents the total access cost of the simulation for varying *RQ size*, *Creation Probability* and *Environment Size* values with their standard deviations for when State Migration (SM) is activated or not.

The cost increases with increasing values of *RQ size* and *Creation Probability*. As the agents range window (sensor range) increases with *RQ size*, the possibility of SSV accesses also increases. Also the likelihood of finding SSVs in the environment increases with *Creation Probability*. We can also observe the impact of combinations of both *RQ size* and *Creation Probability* parameters on Access Cost With SM. When *RQ size* and *Creation Probability* values are set to minimum, a minimal number of SSVs are accessed and mostly agents update their own positions. So, the With SM cost is reduced significantly with localised SSV access. But as *Creation Probability* values increase (keeping *RQ size* values minimum), the number of SSVs accessed increases with the cost as the range window is not large enough to localise SSV access. As both parameters are increased this effect is diminished. However, with the combination of *Environment Size* parameter values, the overall cost seems to decrease with increasing environment size as it reduces the likelihood of locating SSVs in the environment.

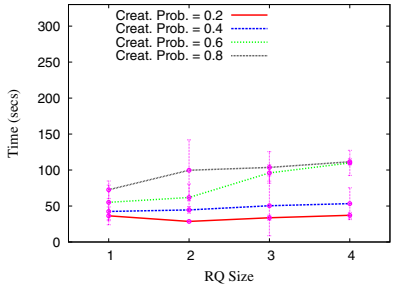
In tileworld simulation, the area polled by agents will change constantly (almost every step) and this is reflected in the propagation values. With SM, overall range query propagation is expected to reduce. Indeed, as agents update themselves and SSVs within their sensor range (more often), this localised access results to an average hop number of 2.8 (compared with 7 without SM). This is illustrated in figure 6a which presents an average update propagation for varying values of *RQ size* and *Environment Size* and variance over different *Creation Probability* values. Increasing the range window decreases the propagation with its deviation. Conversely, figure 6b presents the average update propagation for varying *Creation Probability* and *Environment Size* values and variance over different values of *RQ size*. It shows that increasing the density of the model increases the propagation with its deviation.



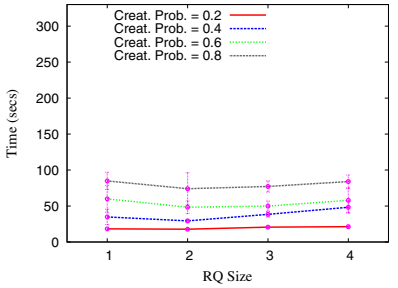
(a) Average Simulation Time without State Migration for *Environment Size* = 25.



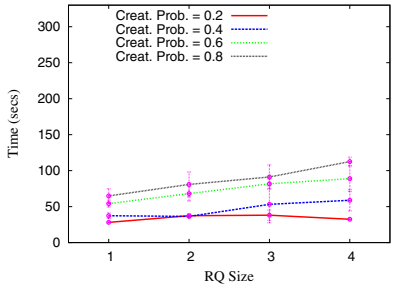
(b) Average Simulation Time with State Migration for *Environment Size* = 25.



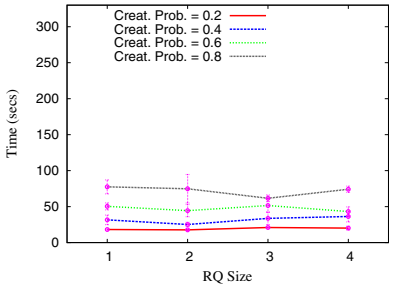
(c) Average Simulation Time without State Migration for *Environment Size* = 50.



(d) Average Simulation Time with State Migration for *Environment Size* = 50.



(e) Average Simulation Time without State Migration for *Environment Size* = 65.



(f) Average Simulation Time with State Migration for *Environment Size* = 65.

Fig. 7: Average Simulation Time for varying *RQ size* and *Creation Probability* with their standard deviations.

4.1.2. Simulation Time

The results presented in this section are meant to analyse the impact of initiating State Migrations (SM) on simulation time (the total elapsed wall clock time to finish an experimental run). Thus far, observing the patterns of access cost and propagation of SSV accesses, we would expect a linear relation with the simulation time. This is indeed observed in figure 7 which presents the average simulation time for varying *RQ size*, *Creation Probability* and *Environment Size* values with their standard deviations. As expected, smaller values of *Environment Size*, increasing *RQ size* and *Creation Probability* increase SSV accesses and, as a result, number of migrations.

4.2. Scalability Analysis

Having analysed the interplay of different workload parameters on the performance of the system, this section focuses on the scalability of PDES-MAS for varying Number of ALPs and Number of CLPs (depth of the PDES-MAS tree). As Number of ALPs increases, the volume of data access in the CLP tree increases too. Following the philosophy of PDES-MAS, the expectation is that for a certain Number of ALPs distributing the SSVs across the CLP tree would increase the performance of the system since migrating state reduces access costs and

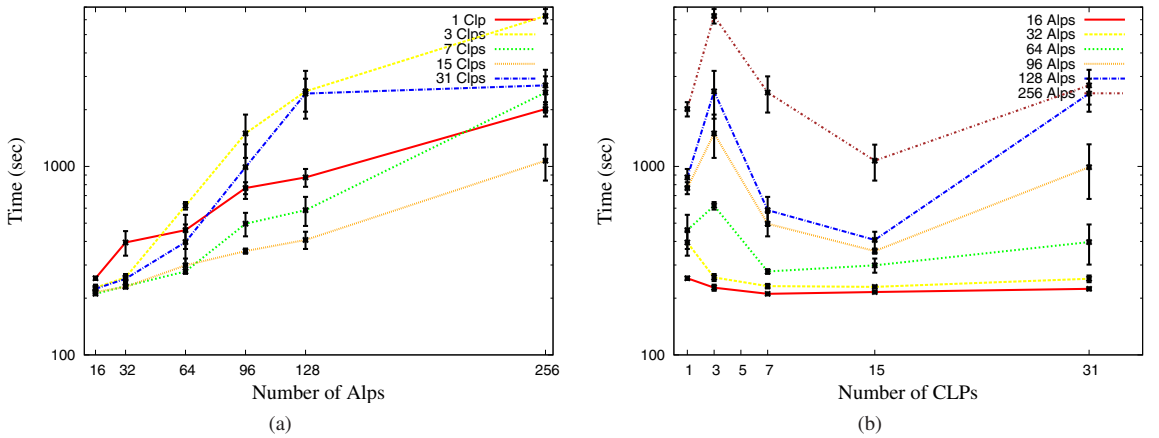


Fig. 8: Average Simulation Time for varying Number of ALPs and Number of CLPs with their standard deviations.

rollbacks. The extra time incurred by rollbacks (due to range updates and state migrations) will be compensated with localised data access. However, as Number of CLPs increases even further we expect a cut off point after which communication takes over, overheads increase and performance degrades. Figure 8 presents the average simulation time for varying Number of ALPs and Number of CLPs with their standard deviations.

Figure 8a shows that simulation time increases with increasing Number of ALPs in the tree. With Number of CLPs = 1 (where dynamic distribution is disabled), the simulation time increases from 254.95 (with Number of ALPs = 16) to 2015.4 (with Number of ALPs = 256). With 3 CLPs, simulation time is much higher as the distribution of the SSV load is not enough to amortise the communication and management overhead introduced by SM. The same phenomenon is observed for 7 CLPs and 31 CLPs (much sooner) where SSVs become too thinly distributed across the CLP-tree making the system communication bound. A tree of 15 CLPs is obviously the optimal configuration for this particular workload set up.

Similar patterns are observed in figure 8b. For a small number of ALPs, moving from 1 CLP to 7 CLPs reduces simulation time which then remains stable - suggesting that SSV accesses are totally localised. For higher number of ALPs, going from 1 to 3 CLPs increases simulation time as the SSV distribution is not enough to amortise the communication and management overheads introduced. After that, we observe a speedup for increasing tree depth up to a point after which communication overhead takes over increasing the simulation time. Clearly, the higher the number of ALPs, the longer it takes before the system becomes communication bound.

5. Conclusions

This paper has presented PDES-MAS, a distributed simulation engine for large scale multi-agent simulations. The system uses a space-time DSM approach to adaptively and dynamically partition the shared state of the MAS and addresses the problems of load balancing, synchronisation and interest management in an integrated transparent manner. The paper has presented a comprehensive quantitative analysis of PDES-MAS and has confirmed the viability of this approach showing substantial gains in access costs and simulation performance.

MAS simulations are increasingly recognised as a key paradigm for deep Big Data Analytics [20, 21]. As we are moving to the exascale computing era, there is a pressing need to support extreme-scale MAS simulations which will be accessing exascale historical and streaming data [22, 23]. Traditional distributed simulation approaches are not suitable for MAS simulations. In order to meet the performance and scalability requirements of this new generation of data-intensive analytics new generation simulation platforms (software and hardware) are required. The work presented in this paper aspires to contribute to this challenging endeavour. Future work will investigate alternative graph structures for the CLPs and the use of proxy variables of SSVs to minimise communication even further.

References

[1] B. Logan, G. Theodoropoulos, The Distributed Simulation of Multi-Agent Systems, Vol. 89, 2001, pp. 174–186.

[2] G. Theodoropoulos, R. Minson, R. Ewald, M. Lees, Multi-Agent Systems: Simulation and Applications, CRC Press, 2009, Ch. Simulation Engines for Multi-Agent Systems, pp. 77–105.

[3] IEEE, IEEE Standard for Modeling and Simulation High Level Architecture (HLA) - Framework and Rules, IEEE Std. 1516-2000 (2000).

[4] IEEE, IEEE Standard for Modeling and Simulation High Level Architecture (HLA) - Object Model Template, http://hla.dmso.mil/hla/tech/omtspec/omt1-3d4.doc., IEEE Std. 1516.2-2000 (2000).

[5] M. Lees, B. Logan, G. Theodoropoulos, Adaptive optimistic synchronisation for multi-agent simulation, in: D. Al-Dabass (Ed.), Proceedings of the 17th European Simulation Multiconference (ESM 2003), Society for Modelling and Simulation International and Arbeitsgemeinschaft Simulation, Society for Modelling and Simulation International, Delft, 2003, pp. 77–82.

[6] B. Craenen, G. Theodoropoulos, V. Suryanarayanan, V. Gaffney, P. Murgatroyd, J. Haldon, Medieval military logistics: a case for distributed agent-based simulation, in: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, SIMUTools '10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Torremolinos, Malaga, Spain, March 21, 2010, pp. 6:1–6:8.

[7] M. Lees, B. Logan, G. Theodoropoulos, Analysing probabilistically constrained optimism, Vol. 21, 2009, pp. 1467–1482.

[8] M. Lees, B. Logan, G. Theodoropoulos, Using access patterns to analyze the performance of optimistic synchronization algorithms in simulations of mas, Vol. 84, 2008, pp. 481–492.

[9] M. Lees, B. Logan, G. Theodoropoulos, Time windows in multi-agent distributed simulation, in: Proceedings of the 5th EUROSIM Congress on Modelling and Simulation (EuroSim'04), Paris, 2004.

[10] V. Suryanarayanan, R. Minson, G. K. Theodoropoulus, Synchronised range queries, in: Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 41–47.

[11] B. Samadi, Distributed simulation, algorithms and performance analysis (load balancing, distributed processing), Ph.D. thesis, Computer Science Department (1985).

[12] F. Mattern, Efficient algorithms for distributed snapshots and global virtual time approximation, Journal of Parallel Distributed Computing 18 (4) (1993) 423–434.

[13] Y. bing Lin, E. D. Lazowska, Determining the global virtual time in a distributed simulation, in: International Conference on Parallel Processing, 1990, pp. 201–209.

[14] K. M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1985) 63–75.

[15] D. Chen, R. Ewald, G. K. Theodoropoulos, R. Minson, T. Oguara, M. Lees, B. Logan, A. M. Uhrmacher, Data access in distributed simulations of multi-agent systems, Journal of Systems and Software 81 (12) (2008) 2345–2360.

[16] T. Oguara, D. Chen, G. K. Theodoropoulos, B. Logan, M. Lees, An adaptive load management mechanism for distributed simulation of multi-agent systems, in: Distributed Simulation and Real-Time Applications, 2005, pp. 179–186.

[17] V. Suryanarayanan, B. G. W. Craenen, G. K. Theodoropoulos, Synchronised range queries in distributed simulations of multi-agent systems, in: Proceedings of the 2010 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, DS-RT '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 79–86.

[18] M. E. Pollack, M. Ringuette, Introducing the tileworld: Experimentally evaluating agent architectures, in: In Proceedings of the Eighth National Conference on Artificial Intelligence, 1990, pp. 183–189.

[19] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.

[20] L. Wallis, Big data, analytics, and storytelling, Causalities: Applied Systems Science, Dynamics and Simulationhttp://blog.cause-alities.com.

[21] FuturICT-Proposal, Global earth simulator, http://www.futurict.eu/.

[22] C. M. Macal, M. J. North, Agent-based modeling and simulation for exascale computing, 2008, pp. 34–41, http://www.scidacreview.org/0802/html/abms.html.

[23] C. Kennedy, G. Theodoropoulos, V. Sorge, E. Ferrari, P. Lee, C. Skelcher, Data driven simulation to support model building in the social sciences, Vol. Volume 5/Number 4, Multi Science Publishing Co Ltd, November 07, 2011, pp. 561–582, special Issue on DDAS.