Efficient Direct Agent Interaction in Optimistic Distributed Multi-Agent-System Simulations

Shuyi Chen
SUSTech, Shenzhen, China
chensy8@mail.sustech.edu.cn

Masatoshi Hanai
SUSTech, Shenzhen, China
mhanai@acm.org

Zhengchang Hua
SUSTech, Shenzhen, China
huazc@mail.sustech.edu.cn

Nikos Tziritas
SUSTech, Shenzhen, China
tziritasn@mail.sustech.edu.cn

Georgios Theodoropoulos*
SUSTech, Shenzhen, China
georgios@sustech.edu.cn

ABSTRACT

Agent-to-agent communications is an important operation in multi-agent systems and their simulation. Given the data-centric nature of agent-simulations, direct agent-to-agent communication is generally an orthogonal operation to accessing shared data in the simulation. In distributed multi-agent-system simulations in particular, implementing direct agent-to-agent communication may impose serious performance degradation due to potentially large communication and synchronization overheads. In this paper, we propose an efficient agent-to-agent communication method in the context of optimistic distributed simulation of multi-agent systems. An implementation of the proposed method is demonstrated and quantitatively evaluated through its integration into the PDES-MAS simulation kernel.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent systems**; **Modeling methodologies**; **Distributed simulation**; **Simulation support systems**.

KEYWORDS

Multi Agent Systems; Distributed Simulation; Interaction

ACM Reference Format:
Shuyi Chen, Masatoshi Hanai, Zhengchang Hua, Nikos Tziritas, and Georgios Theodoropoulos. 2020. Efficient Direct Agent Interaction in Optimistic Distributed Multi-Agent-System Simulations. In *Proceedings of the SIGSIM Principles of Advanced Discrete Simulation (SIGSIM-PADS '20), June 15–17, 2020, Miami, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3384441.3395977

1 INTRODUCTION

Recent years have highlighted and showcased Multi-Agent Systems (MAS) as a powerful computational paradigm in a wide range of domains, including multiplayer online gaming, mobile robots controlling, intelligent manufacturing, smart city and sustainability applications, telecommunications and complex systems modelling [4, 5, 10, 12–14].

An agent can be viewed as a self-contained, concurrently executing thread of control that encapsulates some state and communicates with its environment and possibly other agents. A Multi-Agent System is a system that incorporates multiple autonomous and intelligent agents which interact with both their environment and other agents to cooperate in order to perform some task.

This interaction may come in different forms, protocols and semantics. A taxonomy of the different interaction models for agents may be found in [6]. Irrespective of the details of the different schemes, a major distinction can made between direct and indirect interaction models (Figure 1). In the former the agents exchange information directly via an end-to-end message-passing mechanism that regulates the exchange of messages between agents. In this case, the protocols and semantics of interaction are supported and implemented using an Agent Communication Language (ACL) such as FIPA-ACL [15] or KQML [11]. In the case of indirect interaction, the communication takes place implicitly through the environment, which provides the mediation context for the communication through shared objects. In this case, the agent's Perception and Action operations also provide the means for implicit agent-to-agent communication via changing shared state of the simulation (e.g. perception by an agent of the effects of another agent's action). This interaction model for instance, fits nicely with spatially situated agents.

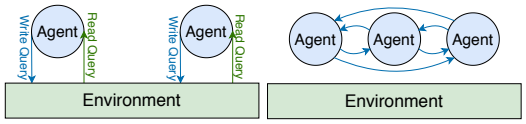

Figure 1: Agent Interaction Models

The complexity of MAS systems renders simulation modelling the only viable method to study their properties and analyse their emergent behaviour. As a result several simulation platforms for MAS have been proposed and developed over the last years. These systems are surveyed in [2]. Furthermore, as MAS models are being applied in ever larger complex System-of-Systems, distributed simulation has emerged as an appropriate approach to deal with their size, scale and complexity. A survey of some of the most representative of them historically has been reported in [19]. A pioneering

*Corresponding author.

SIGSIM-PADS '20, June 15–17, 2020, Miami, FL, USA

ACM ISBN 978-1-4503-7592-4/20/06…$15.00
https://doi.org/10.1145/3384441.3395977

effort in this direction has been PDES-MAS, a distributed simulation framework specifically designed to support large scale MAS models [18]. PDES-MAS provides mechanisms for the adaptive partitioning of the simulation shared state and addresses the problems of load balancing, synchronisation and interest management in an integrated, adaptive and fully transparent manner.

As described in [2], different simulation platforms support different types of MAS models for different domains. Therefore the underlying simulation infrastructure has to provide support for the agent interaction models inherent in the modelled MAS. This should include support for synchronisation coordination, agent perception and discovery, ontological issues etc.

Such support becomes more challenging when it comes to distributed simulation platforms. For direct interaction the infrastructure should supply a reliable end-to-end message passing mechanism while for indirect support for accessing shared data is required. The two communication models, message passing and shared memory, are generally fundamentally orthogonal in distributed systems, so supporting both interaction models in the same distributed infrastructure is not straightforward. In distributed systems, the DSM model (Distributed Shared Memory) has been developed to marry the two communication models. In Distributed Simulation, there have simulation engines with DSM features, that can potentially facilitate the support of both agent interaction models e.g. [9, 16, 18]. However, the need for logical time synchronisation adds another dimension of complexity to the problem as accesses to shared memory and message passing events (i.e. the two respective message streams in Figure 1) need to be put consistently in the same logical time framework.

This paper addresses this challenge in the context of PDES-MAS system. PDES-MAS is based on a space-time Distributed Shared Memory (DSM) model whereby agents interact with their environment and communicate via accessing shared public variables. The agent interaction model supported is the indirect one. Agents can only exchange information through actions on shared objects (write operations) in the environment which can then be perceived by other agents (Read queries) [18]. In this paper we present an approach to exploit the DSM infrastructure of the PDES-MAS to support a direct agent-to-agent interaction model. The fundamental question we address is the following: how can message passing between agents be implemented within a distributed simulation engine that is based on DSM and optimistic synchronisation? The solution we present is based on a mailbox concept.

The rest of the paper is structured as follows: Section 2 provides a brief overview of the PDES-MAS system. Section 3 presents the proposed mailbox method for direct agent-introduction while section 4 discusses its integration in the PDES-MAS system; two methods are discussed, a simple one that is used as a baseline and a more efficient that delivers higher performance. Section 5 presents a quantitative analysis of the proposed method. Section 6 discusses some related work, and section 7 concludes the paper with some thoughts for future work.

2 BACKGROUND: THE PDES-MAS SYSTEM

PDES-MAS is an optimistic distributed agent-based simulation framework and system for large-scale simulation of multi-agent

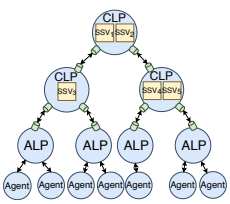

Figure 2: Topology of PDES-MAS

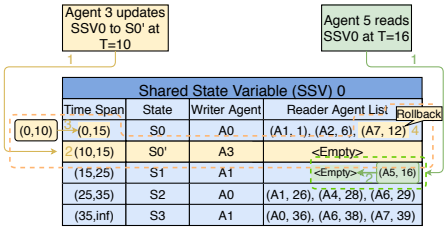

Figure 3: SSV and Read/Write Query.

models. 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 the disjointed state spaces of the model. Two types of LP exist in a PDES-MAS simulation. Agent Logical Processes (ALPs) are responsible for modelling the behaviour of the agents in the MAS. This includes the processing of sense data, the modelling of behavioural processes and the generation of the new actions. ALPs store only private state variables while the shared state, namely public variables referred to as SSVs- Shared State Variables, (including publicly accessible attributes of agents) are distributed over and managed by a tree-like network of server LPs known as Communication Logical Processes (CLPs) (Figure 2). The primary philosophy of PDES-MAS is to provide multiple ALPs concurrent access to a set of SSVs in a scalable manner by adaptive, dynamic reconfiguration of the CLP tree to reflect the interaction patterns between the agents and their environment. Details of the SSV structure and the management of queries in PDES-MAS can be found in [17, 18]. Below we provide a short outline.

The SSV Data Structure: SSV is structured as a list of quadruples as shown in Figure 3. Each quadruple includes (i) timestamp interval from start to end time, (ii) the state variable value for the span, (iii) the id of a writer agent who updates the state, and (iv) ids of reader agents who read the state during the span.

Read and Write Queries to SSVs: Figure 3 also shows how read and write queries are handled in PDES-MAS. A new write period will be inserted when a new write-query arrives (e.g. "Agent 3 updates SSV_0 to S_0' at $T = 10$") , to record the updated value and the agents who read the updated value. The read-query (e.g. "Agent 5 reads SSV_0 at $T = 16$") reads the SSV value for the associated timestamp interval. Then, it adds the reader agent id to the reader agent list, in case the value is invalidated in the future and the reader agents need to rollback.

Rollback Mechanism: PDES-MAS handles rollbacks to recover from conflicts as follows: (i) the conflict occurs at SSV due to a write-query; (ii) the CLP managing the SSV notifies related ALPs to rollback; (iii) the ALP rollbacks its local time and sends anti-messages for queries to the pertinent CLPs which issues further anti-messages to rollback relevant ALPs.

3 PROPOSED ABSTRACT MAILBOX MODEL

The proposed mailbox model for the agent-to-agent communication is outlined in Figure 4 and is designed as an extended form of SSV. In the model, a direct agent-to-agent message (a *mail*) is timestamped with the sending time. Each agent has a mailbox where

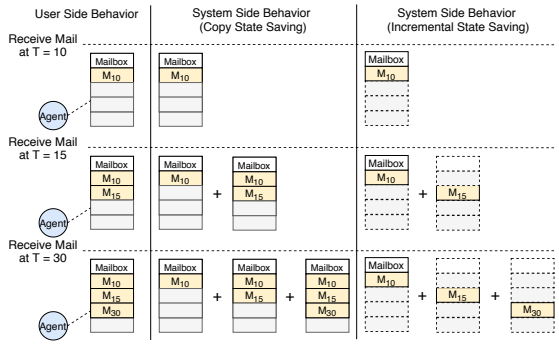

Figure 5: Store History on Native SSV and Mailbox.

received mails are stored in timestamp order and are processed by the corresponding owner agent one by one. The two types of query shown in Figure 4 are read query and write query which are handled through corresponding APIs. A `Mail` class consists of (i) the sending time, (ii) the content and (iii) the id of sender agent. Sending mail is a write query, using `SendMail(Mail)` in `Agent` class. A read query calls `ReadMail(rTime)` in `Agent` class to get the mails received since `rTime` to now.

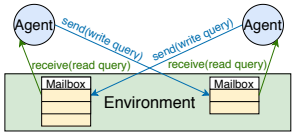

Figure 4: Agent-to-Agent Communication via Mailbox.

4 IMPLEMENTING MAILBOX IN PDES-MAS

In this section, we illustrate an implementation of the mailbox model on top of PDES-MAS. Our key idea utilises (i) the concept of incremental state saving for the compacted data structure and (ii) the existing effective query-handling on SSVs in PDES-MAS. We first discuss a baseline implemenation, which naively uses the original native SSV structures in PDES-MAS. Based on this, we then propose a more efficient method.

4.1 Baseline based on Native SSV

A straightforward way to implement the mailbox mechanism is to use native SSVs as the carrier of mailboxes. Multiple data types are supported by SSVs in PDES-MASUsing this approach, sending a message may be implemented in three steps as follows: (a) a sender agent gets the target-agent's mailbox SSV via a read-query (b) the sender generates a new mailbox which includes the original mails and its new mail appended (c) the sender updates the target-agent's mailbox via write-query.

Similarly, receiving message is implemented as follows (a) a receiver agent gets its own mailbox SSV via read-query (b) the receiver reads the most recent mail of the mailbox (c) the receiver generates a new mailbox with the remaining unread mails in the original mailbox (d) the original mailbox in SSV is updated to the new one via write-query. +

Problems in the Baseline Implementation: Although the baseline method provides a fully compatible implementation for PDES-MAS, it suffers from performance overheads:

(i) Even for simple sending and receiving operations, the framework needs to issue a query twice due to the the fact that in the current implementation of SSVs, incremental updating for an existing value is not supported. (e.g. the appending of new mail to the end of mailbox SSV)

(ii) The baseline implementation does not effectively utilize the efficient read-query handling in PDES-MAS. Both sending and receiving actions include a write-query since the state of the mailbox is modified each time a mail is sent or received.

(iii) Storing the history based on the copy state saving generates a large amount of redundant values. Even though the inserted or dequeued mail changes the state of the mailbox, the remaining mails keep the same value, as illustrated in Figure 5.

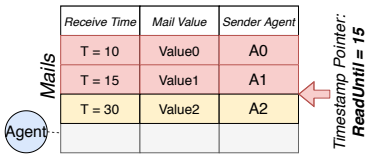

Figure 6: Data Structure of Mailbox.

4.2 Efficient Mailbox Implementation

To address the problems mentioned in the previous section and improve the performance of the system, we propose an efficient mailbox implementation which supports a new type of query, *insert-query* and incremental state saving while fully utilising the read-query optimization in PDES-MAS.

Basic Data Structure of Mailbox: Figure 6 shows the basic data structure for the new mailbox. Each mailbox has a list storing all mails destined for the corresponding agent. Each mail includes the sending timestamp, the sender agent id, and the content.

Instead of storing the entire mailbox as the baseline method does, the new data structure has a timestamp pointer that differentiates unprocessed mails from processed ones. The rollback mechanism can recover a simulation state at a certain time based on the pointer and the stored mails as discussed later in this section. The processed messages are kept until the next GVT synchronisation point when they can be safely removed.

Send/Receive Mail based on Query Processing in PDES-MAS: As we discussed in section 2, PDES-MAS distinguishes two types of queries, read-query and write-query. The key idea of mail processing is to consider the reading of the received mails as the read-query, which can be efficiently handled using a lazy-cancellation-like protocol of PDES-MAS.

The send operation is handled by an extension of the write-query mechanism, referred to as *insert-query*, which allows us to do incremental modification to a specific SSV. By utilizing insert-query, instead of keeping the entire value history for an SSV, the system could save storage space by incrementally storing a transaction for the SSV's modification. The source agent generates an insert-query to the mailbox of the target agent.

With a read operation, the agent gets the most recent mail from its mailbox and processes it (and may undertake some consequent actions). The timestamp pointer of the mailbox will move to the next mail, and the old mail is flagged as processed. This process can be handled as a read-query in PDES-MAS, even though it changes the state of the mailbox (i.e., its timestamp pointer). This is due

to the fact that we can assume that the value of the timestamp pointer never affects the consequent actions of agents. For different timestamp pointers, the agent's action is the same if the received mail is the same. As a matter of fact, our API restricts agents from accessing the timestamp pointer so that agent's action only depends on the received message.

Rollback Mechanism for Mailbox: PDES-MAS makes use of optimistic synchronization, whereby upon the occurrence of a preemption, the rollback mechanism restores the simulation state. For example, for two agents $A1$ and $A2$, $A1$ has already read its received mail at virtual time T_{next} while $A2$ is sending a new mail to $A1$ with receiving logical timestamp T_{prev} ($T_{prev} < T_{next}$). The rollback mechanism restores $A1$'s mailbox state at T_{prev}, and $A1$ reprocess the T_{prev} mail before the T_{next} one. Section 2, outlined how rollbacks are handled in PDES-MAS. More specifically: (a) a preemption occurs when accessing an SSV hosted at a particular CLP; (b) the CLP notifies related agents to rollback; (c) the agent rollbacks and sends anti-messages to other relevant agents; (iv) SSV states are backtracked based on the received anti-messages.

We now extend this mechanism to handle mailbox rollbacks when (i) the conflict occurs at the mailbox, and (iv) the mailbox receives anti-messages. Parts (ii) and (iii) remain the same.

Suppose a preemption case where the local time for the mailbox, T_l, is larger than the insert-query's timestamp, T_q, i.e., $T_l > T_q$. Then, the timestamp pointer of the mailbox moves from T_l to T_q. The managing the mailbox notifies the owner agent to rollback to T_q. The other agents are not involved because the state of the mailbox only affects its owner.

When an anti-message for a mailbox arrives at a CLP, the action depends on the type of query that is being rolled back, as illustrated in Algorithm 1. If the anti-message is caused by a read-query (i.e, $AntiRead(t)$), then the mailbox simply updates its timestamp pointer. If the anti-message is a result of an insert-query (i.e., $AntiInsert(mail, t)$), the mailbox just deletes the message if it is not yet read; otherwise it deletes the message and also notifies the owner agent to rollback.

Algorithm 1: Anti-Message Handling in Mailbox

Input: *AntiMessage* – Anti-message

```
1 if AntiMessage = AntiRead(t) then ReadUntil ← t
2 else if AntiMessage = AntiInsert(mail, t) then
3     if t < ReadUntil then
4         Notify its owner agent to rollback to t
5         ReadUntil ← t; Delete mail at t
6     else Delete mail at t
```

5 EVALUATION

In this section, we evaluate the proposed mailbox approach. We first outline the experimental frame including the benchmark model and the computational platform. We then present the performance results, using the native SSV mailbox implementation as the baseline.

Experimental Frame: For our experiments we use the PDES-MAS platform, which is publicly available at [1] and which we have extended with the mailbox system.

As a benchmark we have developed a MAS model whereby, in the initial stage of simulation, we randomly allocate a fixed number of mail receivers to each agent. The cycle of an agent in this simple model consists of sending mails to agents in the given receiver list continuously and checking new mails regularly. Meanwhile, whenever an agent reads a new mail, it will generate and send a response mail (acknowledgement) to the sender. For simulation of sociable MAS, the initialization of the mailbox communication is controlled by users, which makes it more flexible and customizable. For evaluations of the same model, we use the same initial settings.

The computational environment used in the experiments is a server with two Intel Xeon E5-2697 CPUs ($18 \times 2 = 36$ cores in total) and 512 GB of memory. The operating system is Ubuntu 18.04. For PDES-MAS, we use the GCC 7.4.0 compiler and OpenMPI 2.1.1.

Overall Performance Results: Our analysis compares the two implementations: our proposed Efficient Mailbox (`E-MB`) and the baseline Naive Mailbox (`N-MB`) as a framework to provide insights about the system. The results are shown in Figure 7. To compare the overall performance and memory consumption, we use two metrics: the simulation time and the memory consumption. The simulation time is the total elapsed time for executing the benchmark model. The memory consumption is the summation of all memory used by each mailbox in a specific logical timestamp t.

Our application model simulates different number of agents for 1000 time steps with 31 MPI processes, namely 15 CLPs and 16 ALPs. The numbers of agents are chosen from integer multiples of 64 in [128,1024], with the number of cores fixed at 31. For experiments where the number of cores is the controlled variable, we choose the numbers from power of two minus one within [7, 63], allocate one core per ALP and per CLP and fix the number of agents to 128. Each agent will randomly choose 5 different receiver agents initially, and generate one mail for each receiver per cycle.

Figures 7(a) and 7(b) show the overall simulation time. Figure 7(a) indicates that under the same conditions, the simulation time is significantly reduced by an order of magnitude with the efficient implementation. As the number of agents increases, the trend of growth of the efficient one is more moderate than the growth of naive implementation's simulation time. As illustrated in Figure 7(b), under the same conditions, the simulation time of the application model running on the naive implementation is always longer than the simulation time of the efficient version. As the number of cores increases, the inner structure of the system and the number of LPs are different: more CLPs and ALPs will be added to the system, thus the number of agents carried by one ALP (core) will be smaller. The simulation time of the benchmark model on the naive implementation decreases as there are more available computing resources. However, comparing the curves, it is clear that to execute the same workload, the naive implementation of the mailbox still takes more than ten times longer than the efficient implementation. From the perspective of users, the naive one needs more than 63 cores to meet the performance of the efficient version at 7 cores, which follows that the efficient version of mailbox saves computing resources considerably.

Figures 7(c) and 7(d) show the memory cost for the mailbox. The incremental saving data structure in the efficient implementation optimizes the memory cost of agent-to-agent communication. The

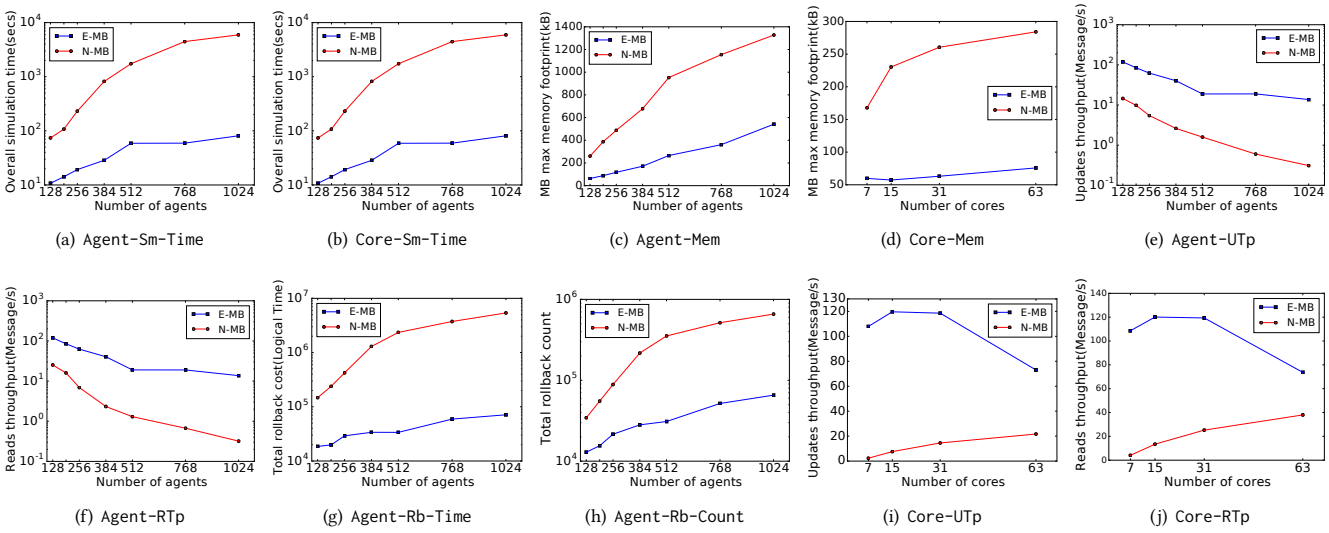

Figure 7: Evaluation Results. (a)–(d) are overall results. (e)–(j) are details.

impact can be observed in Figure 7(c) and Figure 7(d) as the maximum memory cost of mailboxes is less than half of the memory cost of SSV-based implementation under the same environmental conditions. It can also be seen that for the efficient implementation, the growth rate of memory cost with increasing number of agents remains substantially lower than that of the naive implementation. As illustrated in Figure 7(c), we also expect a linear relation between the maximum memory cost and the number of agents involved, which indicates that the communication is scalable.

The maximum memory footprint of the mailboxes against the number of cores is reported in Figure 7(d). When the number of cores increases, there are more computing resources for agents to use, leading to a larger consumption of memory. However, the results for 7, 15, 31 and 63 cores show a similar characteristic: the maximum memory footprint of the efficient version of the mailbox structure is always less than one third than that of the naive implementation. Besides, on the user side, the growth of efficient mailbox memory footprint for increasing cores is small compared with the naive mailbox.

Further Analysis: We delve deeper in the evaluation of the system and analyse four additional metrics: the throughput of message updates, the throughput of message reads, the depth and the number of the rollbacks. The throughput of message updates is the maximum number of messages the agent can add into mailboxes per second. It is calculated by $TPU = 1/t_u$, where t_u is the average time for adding a message to mailbox. The throughput of message reads is the maximum number of messages read by agents per second, following a similar calculation with the message update throughput.

The throughput of updating mailboxes for different number of agents is illustrated in Figure 7(e). Both the writing of a new message or the deleting of a past message lead to the update of the mailbox. From the figure, the downward trend of throughput as the number of agents increases is obvious, which is a result of less computational resources allocated to each agent. It is clear from the figure that the throughput of efficient mailbox remains one order of magnitude higher than that of naive mailbox.

Similar patterns for the throughput of message reads can be observed in Figure 7(f). With the number of agents increasing from 128 to 1024, the same total amount of computing resources are split to more agents, causing the message reads throughput to go down, but the influence of the resource reduction on the naive mailbox implementation is more noticeable than the effect on the efficient one, as the curve of naive mailbox is steeper. Moreover, with the number of agents increasing, the reading throughput of the naive mailbox is an order of magnitude lower than that of the efficient mailbox, and the gap between the two curves becomes increasingly larger. To sum up, under the same conditions, the average message reads capacity of the efficient mailbox is larger than the querying throughput of the naive mailbox, and the difference increases with decreasing resources allocated to each agent, which means that the performance loss of the efficient mailbox is lower under higher message traffic.

With a fixed number of agents and increasing number of cores, the throughput of mailbox updates is shown in Fig 7(i). For the efficient implementation, since state migration is not enabled in the simple application model, with 7 cores the system only contains 3 CLPs and 4 ALPs, so for sending messages to mailboxes or deleting messages in mailboxes, the number of hops required is only two. Therefore, the throughput of mailbox updates is quite large. when the number of cores increases to 15 and 31, there are slight increases in throughput, since the number of hops needed has increased to three and four, respectively. A significant decrease of throughput is shown when the number of cores doubles, and the maximum number of hops increases to 5. The same phenomenon can be observed in the read throughput for the efficient mailbox in Figure 7(j). This is due to the fact that for this tree depth, the system becomes communication bound.

For the update and read throughput of the naive mailbox implementation in Figure 7(i) and Figure 7(j), there is no observable impact of communication overhead, but the throughput mainly

depends on the computational resources each agent can use, since the throughput improves with increasing cores (and, thus, LPs). Despite this, for each pair of points at the same number of cores in the two figures, the throughput of the efficient mailbox is definitely several times better than that of the naive way. Thus, we can conclude that the efficient mailbox implementation significantly improves the communication performance over the naive way as long as the system remains computation bound.

Rollbacks play an important role in distributed simulation based on optimistic synchronization. We have analyzed the total logical time reversed by rollbacks, which relates to the depth of rollbacks. The total rolled back logical time for the simulation of the model is illustrated in Figure 7(g). As the number of the agents increases, agent interactions increase too and thus the total rollback time increases. However, as illustrated in the figure, the rollback cost in the efficient implementation is an order of magnitude less than that of the naive implementation.

The rollback mechanism of the efficient mailbox is also designed with a view to reduce the number of rollbacks for each agent. In the efficient mailbox implementation, actions such as the withdrawing of unread messages or the cancelling of message reads will not cause the rollback of the agents directly, while the naive version of system will rollback all agents involved. Figure 7(h) counts the cumulative number of rollbacks against the number of agents. The figure shows a similar pattern to Figure 7(g). The impact of the efficient mailbox's rollback mechanism on the total number of rollbacks is apparent since the result of the efficient mailboxes remains an order of magnitude less than the result of the naive mailboxes.

6 RELATED WORK

There have been some past efforts to develop direct agent interaction models in distributed simulations. In [20] a general architecture is proposed for HLA (High Level Architecture) utilising local agent proxies to achieve communication. In RepastHPC [8] direct message communication is not supported and communications between agents are carried out by method calls on the target agents. For agents in remote nodes local proxies are created thus introducing consistency problems. In FLAME [7] direct agent communication is achieved through a distributed message board construct. In PANDORA [3] inter-agent communications is limited to local nodes. None of the aforementioned systems support optimistic synchronisation.

7 CONCLUSIONS AND FUTURE WORK

Supporting both direct and indirect agent interaction models in a distributed simulation platform introduces challenging problems related to the need to marry message-passing and shared memory communication within a consistent and coherent logical time synchronisation framework. The work presented in this paper has aspired to address this challenge by proposing a mailbox mechanism for end-to-end agent communication. The robustness and efficiency of the proposed method has been demonstrated through its integration to the PDES-MAS system. In the future we plan to (a) evaluate the mailbox with different and larger scale MAS models (b) support the dynamic migration of the mailbox through the CLP tree, and (c) explore alternative algorithms for direct agent interaction in PDES-MAS that can utilise MPI directly bypassing the PDES-MAS CLP tree.

ACKNOWLEDGEMENTS

This research was supported in part by Guangdong Province Innovative and Entrepreneurial Team Programme, No: 2017ZT07X386; and Shenzhen Peacock Programme, No: Y01276105.

REFERENCES

[1] PDES-MAS. https://pdes-mas.github.io/ (Last access: April 20th, 2020).
[2] S. Abar, G. K. Theodoropoulos, P. Lemarinier, and G. M. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33, 2017.
[3] E. S. Angelotti, E. E. Scalabrin, and B. C. Ávila. Pandora: a multi-agent system using paraconsistent logic. In *Proceedings of Fourth International Conference on Computational Intelligence and Multimedia Applications*, pages 352–356, 2001.
[4] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of 22nd International Conference on Distributed Computing Systems*, pages 15–22, 2002.
[5] M. Balmer, N. Cetin, K. Nagel, and B. Raney. Towards truly agent-based traffic and mobility simulations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 60–67, 2004.
[6] S. Bandini, S. Manzoni, and G. Vizzari. Agent based modeling and simulation: an informatics perspective. *Journal of Artificial Societies and Social Simulation*, 12(4):4, 2009.
[7] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the flame agent-based simulation framework. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pages 538–545, 2012.
[8] N. Collier and M. North. Repast hpc: A platform for large-scale agent-based modeling. *Large-Scale Computing*, pages 81–109, 2012.
[9] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149, 1997.
[10] G. D'Angelo and S. Ferretti. LUNES: Agent-based simulation of p2p systems. In *Proceedings of 2011 International Conference on High Performance Computing & Simulation*, pages 593–599, 2011.
[11] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management*, pages 456–463, 1994.
[12] M. Matskin. Scalable agent-based simulation of players in massively multiplayer online games. In *Proceedings of Eighth Scandinavian Conference on Artificial Intelligence*, volume 3, page 153, 2003.
[13] L. Monostori, J. Váncza, and S. R. Kumara. Agent-based systems for manufacturing. *CIRP annals*, 55(2):697–720, 2006.
[14] J. L. Posadas, J. L. Poza, J. E. Simó, G. Benet, and F. Blanes. Agent-based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*, 21(6):805–823, 2008.
[15] S. Poslad. Specifying protocols for multi-agent systems interaction. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4):15–es, 2007.
[16] M. Principe, T. Tocci, A. Pellegrini, and F. Quaglia. Porting event &cross-state synchronization to the cloud. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 177–188, 2018.
[17] V. Suryanarayanan and G. Theodoropoulos. Synchronised range queries in distributed simulations of multiagent systems. *ACM Transactions on Modeling and Computer Simulation*, 23(4):1–25, 2013.
[18] V. Suryanarayanan, G. Theodoropoulos, and M. Lees. Pdes-mas: Distributed simulation of multi-agent systems. *Procedia Computer Science*, 18:671–681, 2013.
[19] G. Theodoropoulos, R. Minson, R. Ewald, M. Lees, A. M. Uhrmacher, and D. Weyns. Simulation engines for multi-agent systems. *Multi-agent systems: simulation and applications*, pages 77–108, 2007.
[20] F. Wang, S. J. Turner, and L. Wang. Agent communication in distributed simulations. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 11–24, 2004.