Synchronised Range-Queries in Distributed Simulations of Multi-Agent Systems

Journal:	*Transactions on Modeling and Computer Simulation*
Manuscript ID:	TOMACS-2012-0032
Manuscript Type:	Regular Article
Date Submitted by the Author:	25-Feb-2012
Complete List of Authors:	Theodoropoulos, Georgios; IBM Research,
Keywords:	Types of Simulation: Parallel, Types of Simulation: Distributed, Simulation Support Systems, Types of Simulation: Discrete Event

Synchronised Range-Queries in Distributed Simulations of Multi-Agent Systems

VINOTH SURYANARAYANAN, University of Birmingham, UK
GEORGIOS THEODOROPOULOS, IBM Research, Ireland

Range-Queries are an increasingly important associative form of data access encountered in different computational environments including Peer to Peer systems, Wireless communications, Database systems, Distributed Virtual Environments and, more recently, distributed simulations. In this paper, we present and evaluate a system for performing logical-time synchronised Range-Queries over data in the context of distributed simulations of Multi-Agent Systems (MAS). The paper presents algorithms performing instantaneous Queries within an optimistic synchronisation framework and in the presence of dynamic migration of the simulation state. A quantitative evaluation of the effectiveness of the proposed algorithms under different conditions is also presented.

Categories and Subject Descriptors: I.6.5 [**Simulation and Modeling**]: Model Development; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed*; *Parallel*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent Agents*; *Multi-Agent Systems*

General Terms: Design, Algorithms, Performance, Experimentation

Additional Key Words and Phrases: Range-Queries, Data Management, Distributed Simulation, Multi-Agent Systems, Load Balancing, Distributed Virtual Environments

ACM Reference Format:
Suryanarayanan, V., and Theodoropoulos, G. 2012. Synchronised Range-Queries in Distributed Simulations of Multi-Agent Systems. ACM Trans. Mod Comp. Simul.. V, N, Article A (January YYYY), 25 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Very large distributed data structures are used more than ever before in the deployment of distributed applications such as sensor networks, interactive media, Voice-over-IP services, Content Distribution Networks, Peer-to-Peer systems, Distributed Virtual Environments, massively multiplayer online games and distributed simulations. As these applications become larger, more data-intensive and latency-sensitive, scalability becomes a crucial element for their successful deployment. A particular problem that calls for scalable solutions is accessing data. An approach to address the scalability issue in this context is to build systems in a way that the flow of data is optimised to reflect the *interests* of the user population - a paradigm generally referred to as Interest Management (IM), though the precise meaning of ‘interest’ is generally specific to a particular application area. Data access comes in two different forms, namely access via *ID-Queries* and access via *Range-Queries*.

Author’s addresses: Vinoth Suryanarayanan, School of Computer Science, University of Birmingham, UK; Georgios Theodoropoulos, IBM Research, Ireland.
This work was initiated while Georgios Theodoropoulos was a faculty member at the University of Birmingham, UK.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

An ID-Query is taken to mean a read operation which obtains the current value of a data item given its identifier, assumed to be unique in the system. The IM problem in these systems is essentially one of placing and/or replicating shared variables to minimise traffic in the context of concurrent read and write operations from the nodes in the system. An example is cache coherent NUMA approaches which have attempted to reduce the communication load by switching between write-update and write-invalidate consistency models according to access patterns (e.g. [Eggers and Katz 1989]).

A Range-Query is an operation obtaining a set of data items each of whose current value matches a given predicate, expressed as a contiguous range between two values. The semantics of a Range-Query are not formally defined and are different for different systems and applications but in general two different forms may be distinguished:

— **instantaneous Queries** of the set of currently extant data items whose value matches some predicate, and
— **persistent Subscriptions** to all possible future values data items may take which match some predicate.

Range-Queries, with different terminology and semantics, can be encountered in several areas and at different levels in computer systems research. from hardware (e.g. Content Addressable Memories [Pagiamtzis and Sheikholeslami 2006]), to operating systems (e.g. Linda [Gelernter 1985]) and application level (e.g. relational databases research [Pagel et al. 1993]). In the area of Peer-to-Peer systems, a significant body of recent work has looked at ways to extend the fundamentally ID-Query oriented nature of DHTs to efficiently support Range-Query operations utilising hash-based and hash-free approaches. Most work in this area is geared towards Instantaneous Queries (e.g. [Bharambe et al. 2004]).

In traditional distributed simulations, which are based on message passing, the problem of ID-Queries has been addressed through the development of Distributed Shared Memory (DSM) mechanisms [Mehl and Hammes 1993]. More recently, an explosion of application of the PDES paradigm to more non-traditional simulation tasks with Range-Query functional requirements, such as complex system or multi-agent system models, has led to efforts to bridge the gap between traditional PDES and this increasingly relevant, form of simulation model [Logan and Theodoropoulos 2001; Lees et al. 2003]. These agent-based systems may consist of billions of agents [Macal and North 2008] which operate in a complex shared environment wherewith they interact in highly dynamic and unpredictable patterns. This interaction is performed via a recurring cycle of '*sense-think-act*' operations [Logan and Theodoropoulos 2001], wherein a *sense* is semantically a Range-Query. Most of the relevant work in this direction has been conducted in the fields of large scale distributed simulation (HLA) and more generally Distributed Virtual Environments (DVEs). Though DVEs do use ID-Queries for many operations, they are more commonly characterised as systems which interact with the world primarily via Range-Queries.In this context the Subscriptions query model is adopted. HLA's DDM service provides the ability to associate individual simulation nodes ('federates') with subscription regions (equivalent to Range-Queries) and individual updates with update regions (equivalent to writes). In HLA, Cell-based architectures map queries to multicast groups that correspond to discrete cells in a grid overlay [Hook et al. 1994; Boukerche and Roy 2002; Macedonia et al. 1995; Morse 2000; Berrached et al. 1998] while Region-based approaches calculate explicitly the overlaps between regions themselves [Morse and Zyda 2002]. Similarly, several cell-based [Knutsson et al. 2004; Yu and Vuong 2005; Minson and Theodoropoulos 2007], zone-based [Barrus et al. 1996] and aura-based [Morgan et al. 2004; Hu and Liao

2004] IM approaches have been proposed to support Range-Queries in the context of DVEs and Massively-Multiplayer Online games.

This paper discusses the problem of realising instantaneous Range-Queries in the context of PDES-MAS [Logan and Theodoropoulos 2001], a framework and a system for the distributed simulation of multi-agent models. Different aspects of PDES-MAS have been previously reported in a number of publications: issues related to data distribution and load balancing are discussed in [Ewald et al. 2006; Oguara et al. 2005], synchronisation algorithms have been proposed in [Lees et al. 2009; 2008; 2003], while the utilisation of PDES-MAS to address real-world large scale modelling problems is discussed in [Craenen et al. 2011; Craenen et al. 2010]. The algorithms discussed in this paper were first outlined in [Suryanarayanan et al. 2009] and [Suryanarayanan et al. 2010] where an initial evaluation was also presented. This paper presents a detailed and comprehensive description of the algorithms and a complete set of performance results.

The rest of the paper is organised as follows: Section 2 provides a short description of the PDES-MAS system. Section 3 discusses the design of logical-time synchronised Range-Queries in PDES-MAS while section 4 extends this design to accomodate dynamic load balancing. Section 5 provides a experimental analysis of the proposed algorithms. Finally, Section 6 summarises the main conclusions and outlines future work to be carried out in this area.

2. PDES-MAS

The PDES-MAS framework is implemented based on PDES paradigm, where a simulation model is divided into a network of concurrently executing Logical Processes (LPs), each maintaining and processing disjoint state spaces of the system. 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 1a. CLPs essentially implement a DSM model whereby public variables are represented by *Shared-State Variable* (*SSV*) data-structures which store the history of values taken by a particular variable over time [Lees et al. 2003].

In response to the sensing and acting of agents in the simulation, ALPs perform reads and writes on SSVs in the system. The primary philosophy of PDES-MAS is to provide multiple ALPs concurrent access to the 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 data-structures maintaining routing information. The CLP tree is reconfigured dynamically and automatically (see section 4) 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.

A CLP is responsible for synchronising the events it receives from ALPs. Synchronisation algorithms that have been developed for PDES-MAS are reported in [Lees et al. 2009; 2008; 2003]. In general, 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 1b. If a write period is subsequently invalidated by a straggler write, any ALPs which read that period must by rolled back. ALP reads in

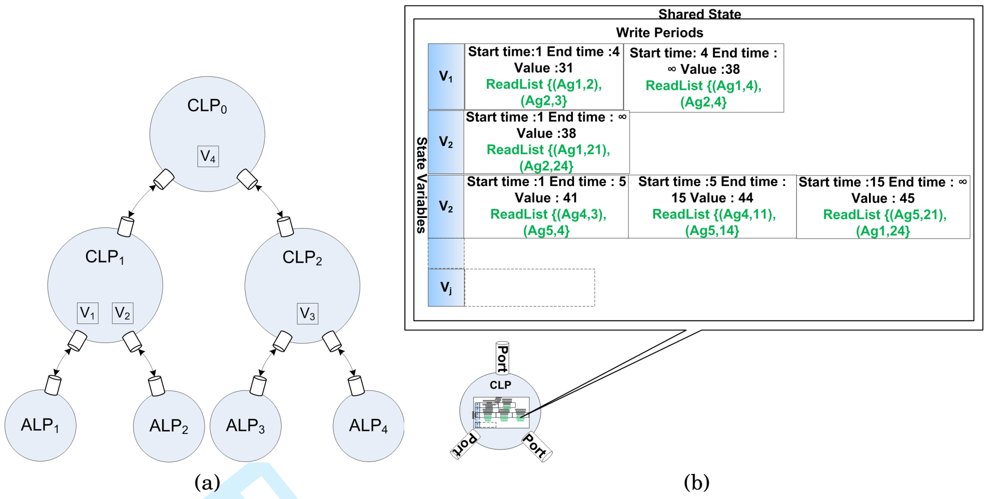

CLP0
V4
CLP1
V1
V2
CLP2
V3
ALP1
ALP2
ALP3
ALP4
(a)
Shared State
Write Periods
State Variables
V1
Start time:1 End time :4 Value :31 ReadList {(Ag1,2), (Ag2,3}
Start time: 4 End time : ∞ Value :38 ReadList {(Ag1,4), (Ag2,4}
V2
Start time :1 End time : ∞ Value :38 ReadList {(Ag1,21), (Ag2,24}
V2
Start time :1 End time : 5 Value : 41 ReadList {(Ag4,3), (Ag5,4}
Start time :5 End time : 15 Value : 44 ReadList {(Ag4,11), (Ag5,14}
Start time :15 End time : ∞ Value : 45 ReadList {(Ag5,21), (Ag1,24}
Vj
Port
CLP
Port
Port
(b)

Fig. 1: A tree of 3 CLPs and 4 ALPs. SSVs within a CLP

PDES-MAS come in two forms: ID-Queries and Range-Queries (sense). The rest of the paper focus on the latter.

3. RANGE-QUERIES IN PDES-MAS

A standard ID-Query read operation in PDES-MAS requests the value of an SSV by specifying the id of the SSV. The kernel handles this by forwarding the message to the CLP hosting the specified SSV. This in turn relies on each CLP maintaining a map from SSV id to forwarding port. In the case of a Range-Query operation, this map is of no use since it records only the ids of SSVs, not their values. Without additional information CLPs would need to flood a Range-Query message to ensure all SSVs are evaluated against it. Different design options for implementing this operation in a more efficient manner were explored in [Ewald et al. 2006]. This paper adopts a logical-time synchronised Range-Based approach.

3.1. Design Outline

The essential idea of the Range-Based system of routing Range-Queries is that each CLP port records the complete value range of all SSVs that can be found beyond it. When a Range-Query is issued it is flooded down all ports such that the port's range overlaps the query's range.

For a given Range-Query this creates a horizon describing the extent of the query's propagation. All CLPs inside the horizon have had all their SSVs scanned by the query, all CLPs outside the horizon claimed to have no SSVs of interest to the query and were therefore not scanned. The horizon itself consists of a set of ports, each of which records the fact that the query was blocked at that point and did not progress. This simple idea is depicted in Figure 4a where the horizon created by a single Range-Query (here, an ALP issuing a Range-Query 2,5) is depicted.

The Range-Query's existence is recorded explicitly at every CLP inside a propagation horizon. Due to this, any changes to SSV values which change the query's results will be detected at the CLP directly. Conversely, CLPs outside the horizon have no record of the query's existence and therefore cannot directly determine if changes to SSV values need to rollback the query or not. Instead, CLPs outside the horizon have

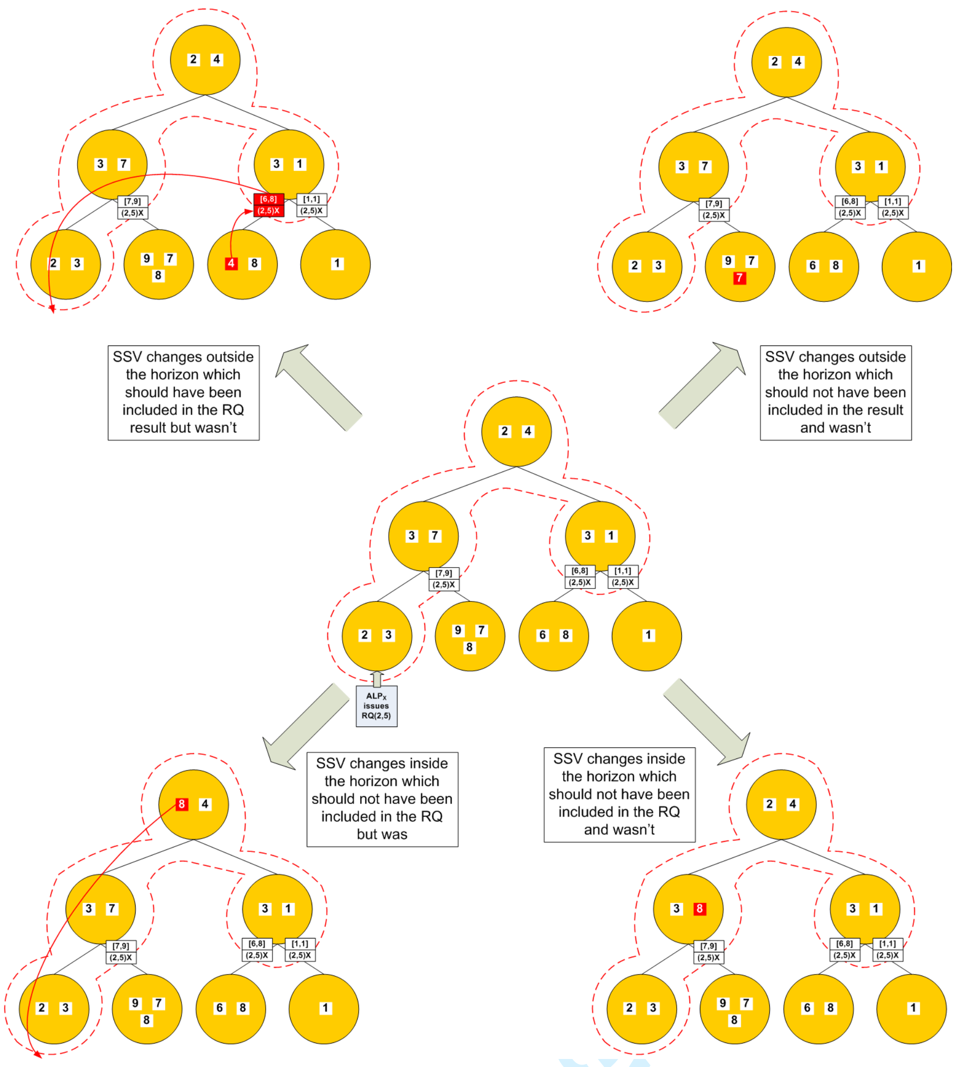

SSV changes outside the horizon which should have been included in the RQ result but wasn't
SSV changes outside the horizon which should not have been included in the result and wasn't
ALPX issues RQ(2,5)
SSV changes inside the horizon which should not have been included in the RQ but was
SSV changes inside the horizon which should not have been included in the RQ and wasn't
[7,9] (2,5)X
[6,8] (2,5)X
[1,1] (2,5)X

Fig. 2: Reactions to SSV changes inside and outside of a propagation-horizon. Changes inside are directly detected by the SSVs the query accessed. Changes outside are detected by range-updates to ports lying on the horizon.

the responsibility to update the port ranges of neighbouring CLPs. In turn, if a port lies on a horizon, it has the task of detecting whether a range update invalidates the earlier decision not to propagate the query and, if so, rollback the entire Range-Query operation.

These two rollback conditions (an SSV change inside and horizon and an SSV change outside the horizon causing a port-range change on the horizon) are shown being evaluated and committed in Figure 2.

3.2. Logical-Time Range Updates

The system described in the above section is relatively simple. Each CLP simply needs to maintain the range covered by its own SSVs and, whenever they change, determine whether port updates need to be sent to neighbouring CLPs. This requires a simple test: is the (min,max) of the SSV set the same as old (min,max)? In turn these changes will be propagated by neighbouring CLPs if they lead to further ports being invalidated.

However, this simple picture is complicated by the fact that an SSV is not a single value, it is a list of time periods during which the SSV took different values. Correspondingly, the set of SSVs maintained by a CLP describe a sequence of ranges over time. It is therefore not correct to think of a port as having information about a range covered by the SSVs beyond the port, but actually a sequence of Range Periods.

In order to maintain the Range Periods of a neighbouring CLP's port we require a more complex algorithm than the simple (min,max) comparison. Before defining this algorithm we first define some data structures and terminology:

— A RangePeriod (RP) describes a period, starting at a given Logical Time (LT) during which a given set of SSVs fall within the specified range.
— A RangeList is a list of RPs which together describe the coverage of the set of SSVs as it evolves over time.
— Each CLP contains a single RangeList H (denoting 'here') for its own local SSVs and has the responsibility of maintaining this data structure.
— In addition, each port to a neighbouring CLP is labeled with a RangeList covering the set of SSVs in *all* CLPs beyond this port. These three port RangeLists are termed U, R and L ('up', 'left' and 'right') respectively. It is the responsibility of the neighbouring CLP to which a port connects to keep this data structure updated.

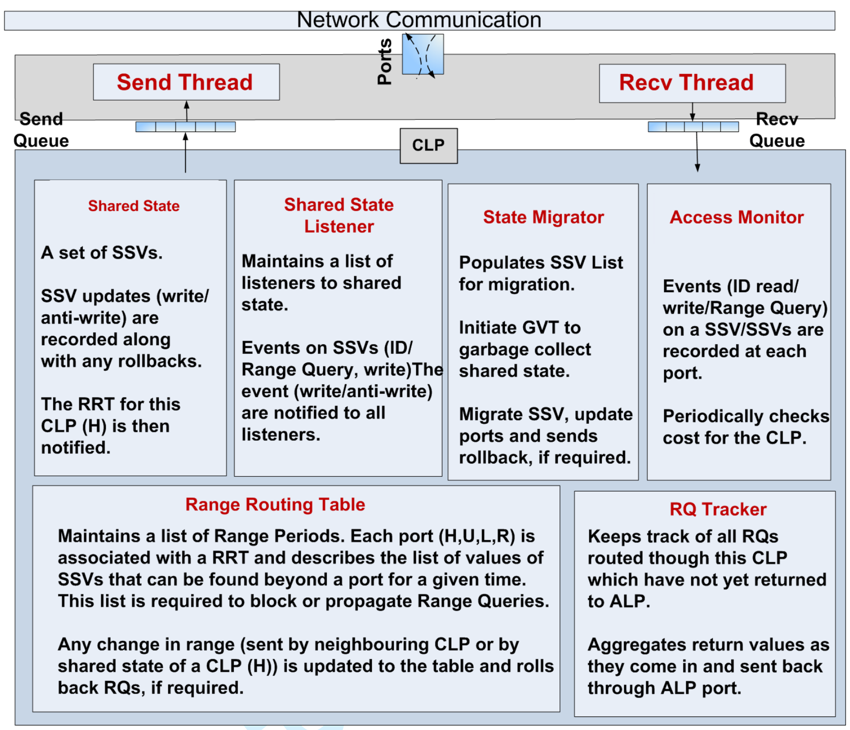
Given these definitions the tasks of the system break down in to three concurrent processes:

(1) As write and anti-write messages are received by the CLP, the RangeList H must be maintained. When H changes, the port RangeLists U, R and L must also be checked and, if necessary, updated with these updated propagated to further CLPs.
(2) As Range-Queries arrive, the RangeLists of the CLP must be used to propagate the query to CLPs with matching SSVs, block it from CLPs without matching SSVs and determine whether the query needs to read local SSVs.
(3) As RangeLists are updated (either via Range Updates from neighbouring CLPs, or by write/anti-write messages bound for the local SSV set) Range-Queries which were blocked at the time they arrived must be checked to determine whether this decision was correct or not. In the event that it was not, the query must be rolled back.

There are three main CLP components involved handling Range-Queries, updates and rollbacks in a time synchronised manner namely, *Range Routing Table (RRT), RQTracker* and *SharedStateListener*(figure 3). The remainder of the section discusses these processes in more detail.

3.2.1. Maintaining Range Period List. The Range Routing Table (RRT) maintains a list of range periods at each port (H, U, L, R) and sends updates to neighbouring CLPs (triggered indirectly by neighbouring CLPs or by the shared state of a CLP (H)). A Range period list at a port (H, U, L, R) represents a list of Range Periods evolved over time. A Range Period is similar to a Write Period which defines the validity of a value of an SSV within a time period (a start and end time). In case of a Range Period, it represents a range of values of a set of SSVs covered during a time period. A simple example of a Range Period List maintained in a CLP is depicted in the figure 4b.

Any change in the value of an SSV is indicated to RRT maintaining Range Period List at port H using the Shared State Listener module. This is an event listener on



Shared State

A set of SSVs.

SSV updates (write/anti-write) are recorded along with any rollbacks.

The RRT for this CLP (H) is then notified.

Shared State Listener

Maintains a list of listeners to shared state.

Events on SSVs (ID/Range Query, write)The event (write/anti-write) are notified to all listeners.

State Migrator

Populates SSV List for migration.

Initiate GVT to garbage collect shared state.

Migrate SSV, update ports and sends rollback, if required.

Access Monitor

Events (ID read/write/Range Query) on a SSV/SSVs are recorded at each port.

Periodically checks cost for the CLP.

Range Routing Table

Maintains a list of Range Periods. Each port (H,U,L,R) is associated with a RRT and describes the list of values of SSVs that can be found beyond a port for a given time. This list is required to block or propagate Range Queries.

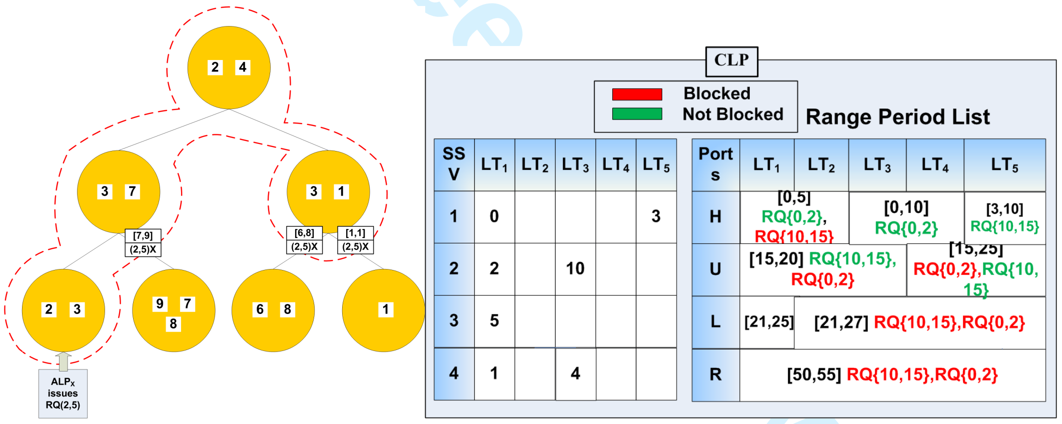
Any change in range (sent by neighbouring CLP or by shared state of a CLP (H)) is updated to the table and rolls back RQs, if required.

RQ Tracker

Keeps track of all RQs routed though this CLP which have not yet returned to ALP.

Aggregates return values as they come in and sent back through ALP port.

Fig. 3: An overview of CLP components relevant to handling Range-Queries.

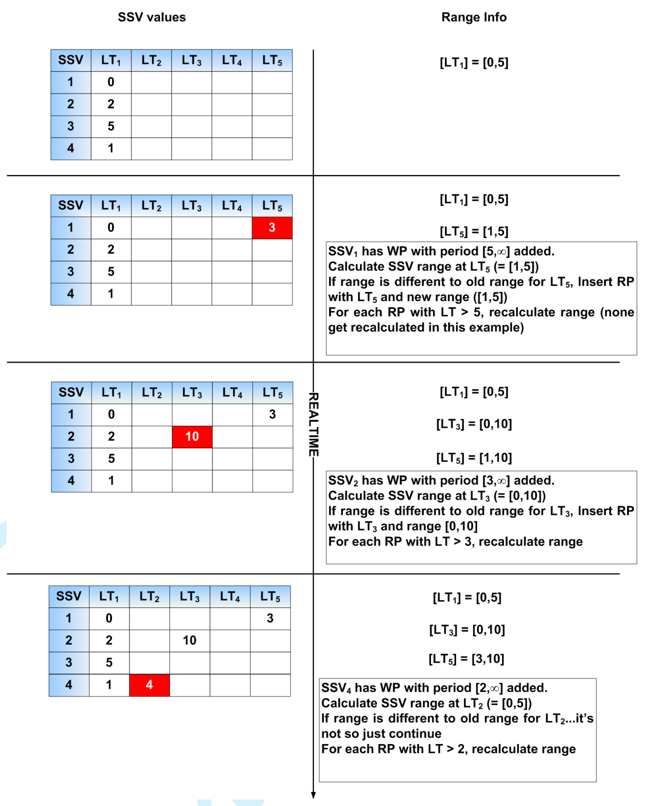


SSV	LT_1	LT_2	LT_3	LT_4	LT_5
1	0				3
2	2		10		
3	5				
4	1		4		

Ports	LT_1	LT_2	LT_3	LT_4	LT_5
H	[0,5] RQ{0,2}, RQ{10,15}		[0,10] RQ{0,2}		[3,10] RQ{10,15}
U	[15,20] RQ{10,15}, RQ{0,2}			[15,25] RQ{0,2},RQ{10,15}	
L	[21,25]	[21,27] RQ{10,15},RQ{0,2}			
R	[50,55] RQ{10,15},RQ{0,2}				

Fig. 4: (a) The propagation-horizon created by a Range-Query traversing a tree of CLPs. (b) An example of a Range Period List (at the right) of a CLP at ports (H, L, U, R) evolved over time.

the shared state of a CLP at port H. All events are indicated automatically to objects registered with this module such as Range Period List at port H and *Access Monitor* (which monitors the access patterns on SSVs, see section 4). In the case of maintaining the Range Period List, only write/anti-write events are indicated. The process for



SSV	LT$_1$	LT$_2$	LT$_3$	LT$_4$	LT$_5$
1	0				
2	2				
3	5				
4	1				

[LT$_1$] = [0,5]

SSV	LT$_1$	LT$_2$	LT$_3$	LT$_4$	LT$_5$
1	0				3
2	2				
3	5				
4	1				

[LT$_1$] = [0,5]

[LT$_5$] = [1,5]

SSV$_1$ has WP with period [5,∞] added.
Calculate SSV range at LT$_5$ (= [1,5])
If range is different to old range for LT$_5$, Insert RP with LT$_5$ and new range ([1,5])
For each RP with LT > 5, recalculate range (none get recalculated in this example)

SSV	LT$_1$	LT$_2$	LT$_3$	LT$_4$	LT$_5$
1	0				3
2	2		10		
3	5				
4	1				

[LT$_1$] = [0,5]

[LT$_3$] = [0,10]

[LT$_5$] = [1,10]

SSV$_2$ has WP with period [3,∞] added.
Calculate SSV range at LT$_3$ (= [0,10])
If range is different to old range for LT$_3$, Insert RP with LT$_3$ and range [0,10]
For each RP with LT > 3, recalculate range

SSV	LT$_1$	LT$_2$	LT$_3$	LT$_4$	LT$_5$
1	0				3
2	2		10		
3	5				
4	1	4			

[LT$_1$] = [0,5]

[LT$_3$] = [0,10]

[LT$_5$] = [3,10]

SSV$_4$ has WP with period [2,∞] added.
Calculate SSV range at LT$_2$ (= [0,5])
If range is different to old range for LT$_2$...it's not so just continue
For each RP with LT > 2, recalculate range

Fig. 5: A sequence of updates occurring to a set of SSVs in real time (top to bottom) and the updates made to the RangePeriodList in response. For each update the algorithm execution is detailed.

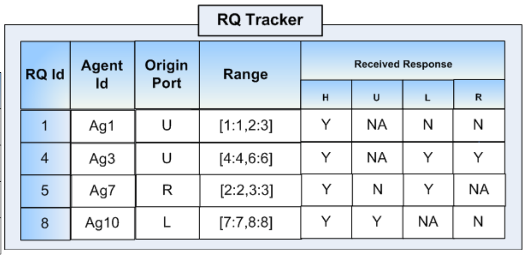
maintaining the Range Period List data structure is illustrated for a sequence of updates to an example SSV set in figure 5. This process in detailed in algorithm 1 in the appendix. This algorithm can be optimised using heuristics but for clarity is presented in its brute-force form. It has the effect of recalculating the entire RangePeriodList beyond the write/anti-write logical time.

3.2.2. Range Update Propagation. Whenever an invocation of the algorithm 1 results in a change to the RangePeriodList H of a CLP, this information may effect the RangePeriodLists associated with ports with neighbouring CLPs. Therefore, whenever a change to H occurs, this information is checked for correctness, and updated if necessary. Of course, the ports U,R and L are not simply maintained with the range of H, rather each is maintained with the union of H and the other two ports (e.g. the range of L is equal to $[min(U_{min}, R_{min}, H_{min}), max(U_{max}, R_{max}, H_{max})]$).

Write and anti-write messages may cause changes to H, leading all ports to be recalculated for correctness with the logic above. Consequently Range Update messages may be sent, leading to the modification of RangePeriodLists at neighbouring CLPs, this in turn will lead to the recalculation of the other ports at that CLP. This relationship between list modifications and checks is depicted in the matrix in figure 6a. The algorithm to handle synchronised range updates arriving at any port (H, U, L, R) is illustrated in the appendix (algorithm 2).

List Modified	Lists Recalculated (using)
H	L(URH) R(HLU) U(RHL)
L	R(HLU) U(RHL)
R	L(URH) U(RHL)
U	L(URH) R(HLU)

(a)



RQ Tracker

RQ Id	Agent Id	Origin Port	Range	Received Response			
				H	U	L	R
1	Ag1	U	[1:1,2:3]	Y	NA	N	N
4	Ag3	U	[4:4,6:6]	Y	NA	Y	Y
5	Ag7	R	[2:2,3:3]	Y	N	Y	NA
8	Ag10	L	[7:7,8:8]	Y	Y	NA	N

(b)

Fig. 6: (a) The relationship between RangePeriodList modification and recalculation. (b) A snapshot of RQTracker table.

3.2.3. Range-Query Processing. This section presents mechanisms for a synchronised Range-Query to propagate, record its existence and return responses from other CLPs back to the originating port. A simple example of a Range Period List with a list of Range-Queries processed at each port of a CLP is depicted in figure 4b. When a Range-Query arrives it is processed by determining which, if any, of the 4 potential sets of SSVs it is required to read: the local set, bounded by H and each of the sets past the port Range Period Lists $\{U, R, L\}$.

Each of the Range Period Lists are read in turn for the given logical time of the query. If the Range Period at that logical time has a non-empty intersection with the query then the query is forwarded to this SSV set (in the case of H this involves reading all local SSVs, in the case of U,R and L this involves the CLP forwarding the message to the neighbours on those ports). In this case, the Range-Query is marked as 'Not Blocked' at each port recording its propagation. If conversely the Range Period has an empty intersection with the query then the query is recorded in the period and is not forwarded to the SSV set. In this case, the Range-Query is marked as 'Blocked' recording its end of propagation extent. This pattern of port traversal, blocking and SSV reading creates the propagation horizon depicted previously in figure 4a.

Each CLP is also responsible to aggregate responses of a Range-Query from ports through which it has been propagated. *RQTracker* table maintains a record of each Range-Query with its unique identifier, originated port (the port the query came from) and status of responses received from ports it has been propagated. A snapshot of RQTracker table is depicted in figure 6b. Once results from all ports are aggregated they are sent back to the originated port and the entry is deleted.

3.2.4. Rollback Control. In PDES-MAS a rollback must be triggered when an ALP is found to have read in incorrect value. This occurs, in the case of ID-queries, when a WritePeriod containing a read with logical time T_R is modified with a new write occuring at time $T_W < T_R$. But in the case of Range-Queries, it is more complicated for two reasons,

— A Range-Query adds a second predicate to the rollback condition. As before a rollback condition will only occur if the new write time T_W is less than the Range-Query time T_{RQ}. However, because an ALP does not need to be rolled back if it did not receive incorrect data, we must also determine if a Range-Query actually returned erroneous data from the WritePeriod or not. This will be true in all cases with the exception of when the WritePeriod value at T_{RQ} was outside the query's range and the value of the write V_W is also outside the query's range. In this case (as depicted in the right hand side of Figure 2) the query was answered correctly despite the new write and does not need to be rolled back.

— A Range-Query will be rolled back in response to straggler/late range updates from neighbouring CLPs using Range Period List. Range Period List records the existence of Range-Queries at all ports, as shown in section 3.2.3, which in turn evaluates the range updates arriving at a port against Range-Queries blocked at that port. Suppose a range update arriving at time T_{RU} less than any of the blocked Range-Queries time T_{RQ} and with range value intersecting any of the blocked Range-Queries' window, then it will be rolled back. This is turn invalidates the query answer because a blocked Range-Query should now be propagated to get new set of SSVs.

Although a Range Period List and a list of Write Periods are conceptually and functionally different entities in the system, algorithmically they share a common approach to storing Range-Queries and processing rollback predicates in response to updates. This process is shown in algorithm 3 in the appendix. .

4. RANGE-QUERIES IN THE PRESENCE OF STATE MIGRATION

The algorithm presented so far assumed fixed locations of SSVs in the tree and did not consider the dynamic reconfiguration of the tree. In this section we focus on the latter. The hypothesis is that the reconfiguration of the tree results at the dynamic reduction of the horizon size of the Range-Queries travelling over the CLP tree.

There are different ways to reconfigure the CLP tree, from splitting and merging CLPs, to migrating ALPs and/or SSVs through the tree [Logan and Theodoropoulos 2001]. Here, we assume a fixed tree of CLPs with ALPs as leaves in fixed locations and SSVs migrating through the tree closer to the ALPs that access them. SSV migration is achieved by means of a competitive optimisation algorithm reported at [Oguara et al. 2005]. Following this approach, a CLP monitors the access patterns of its SSVs at each port maintaining access cost information for each SSV it hosts. The total access cost of an SSV is a function of the number of accesses and the number of hops for each access. SSVs are migrated through the port with the highest access cost over a period. Several threshold parameters are utilised to avoid thrashing and ensure a balanced load across CLPs (see figure 3). This basic algorithm is extended to support Range-Queries.

Migration of an SSV as achieved by deleting the SSV from one CLP and adding to another CLP together with its entire history of *Write Periods*. The ports of both CLPs may be easily updated with the SSV's new location. This does not affect processing any transient simulation messages such as ID Query/write operations.

To support Range-Queries however, migrating an SSV with its Write Period history also means changing the Range Period history at both CLPs (namely the validity periods of range information of SSVs within a CLP). To achieve this, each Range Period has to be evaluated against the list of SSVs selected for migration, again with its entire history. This is because each write period of an SSV could contribute to one or more Range Periods (depending on the validity period of the value). The migration algorithm within each CLP is outlined in the following steps:

(1) *Populate SSV List*: Generate lists of SSVs and the ports to which they have to be migrated.
(2) *Initiate Global Virtual Time*: Whenever the migration algorithm selects an SSV, it first initiates Global Virtual Time (GVT) to garbage collect the shared state from the simulation. The objective of this operation is to reduce the overhead of evaluating the Range Periods. Since every action is timestamped and recorded for rollbacks, it has to be ensured that the SSVs are no longer required for any LP in future. For the work presented in this paper, Mattern's GVT algorithm [Mat-

tern 1993] is used in PDES-MAS to calculate the global minimum time of all send, receive and transient events over all LPs.

(3) *Delete SSVs from the CLP*: Upon completion of the GVT operation, the SSVs selected for migration are deleted. Deletion of a SSV with a list of write periods from a CLP is implemented as a list of anti-writes/rollbacks. An anti-write/rollback operation also handles evaluating the Range Periods generating rollbacks and range updates to neighbouring CLPs.

(4) *Add SSVs to the destination CLP*: The SSVs are migrated to the appropriate destination/neighbouring CLP. The receiving CLP accepts the SSV and adds it and its list of write periods. Adding an SSV to a CLP is implemented as a list of write operations. The write operation also handles evaluating the Range Periods and generating rollbacks and range updates to neighbouring CLPs.

The migration process is depicted in algorithm 4 in the appendix while figure 7 illustrates the operation of the algorithm, showing the state of a CLP after the GVT operation, and the deletion and addition of an SSV

The advantage of coinciding GVT calculation with SSV migration is that the migration process will not affect any transient messages in the simulation and that Range-Queries over the CLPs and Range Updates are synchronised correctly. However, an increased overhead is generated in the form of additional rollbacks and range updates.

5. EXPERIMENTAL INVESTIGATION

In this section we present an analysis of the proposed algorithms using traces generated from different agent-based simulations. The experiments have been designed to analyse the behaviour and performance of the algorithms over various metrics and under different conditions. Two sets of experiments were performed. The first set (section 5.2) is meant to isolate and analyse the behaviour of the Range-Query system without the added complexity of tree reconfiguration. The second set then performs a more comprehensive analysis of the system taking into account dynamic reconfiguration of the system (section 5.3).

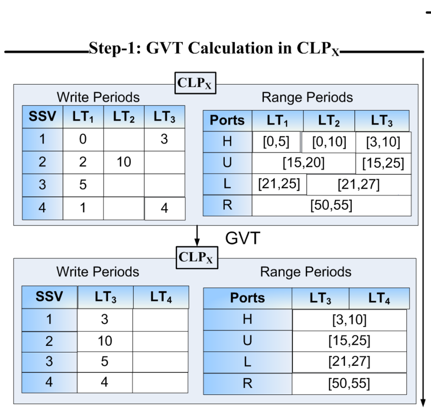
5.1. Experimental Setup

The experiments were carried out on a working implementation of the PDES-MAS kernel written in C++ using simulation traces to control the read, write and Range-Query operations issued by ALPs. The traces used were generated by varying the two experimental parameters: the size of the query (*RQ size*) and the range of values covered by each CLP (*SSV range*), both both proportional to the size of 2-dimensional area covered by the simulation. Figure 8 shows a logical view of the simulation setup for different values of the experimental parameters. Varying the two experimental parameters will have an effect on the experimental behaviour of PDES-MAS. The experimental setup used for analysis presented in section 5.2 is given below.

— A tree of 7 CLPs are initialised, each maintaining 10 SSVs whose values (x, y) lie in a 2-Dimensional radius around a randomly selected central point. The size of the radius corresponds to the *SSV range* parameter.

— 8 ALPs (two attached to each leaf CLP) each issue a 2-Dimensional Range-Query once per tick with a static position and radius. The radius corresponds to the *RQ size* parameter.

A similar setup has been used for the analysis presented in 5.3 but with the root CLP containing initially 7 clusters of 50 SSVs. All SSVs residing at the root CLP represents the average case. Simulations are run for 1000 ticks.



Write Periods

SSV	LT_1	LT_2	LT_3
1	0		3
2	2	10	
3	5		
4	1		4

Range Periods

Ports	LT_1	LT_2	LT_3
H	[0,5]	[0,10]	[3,10]
U	[15,20]		[15,25]
L	[21,25]	[21,27]	
R	[50,55]		

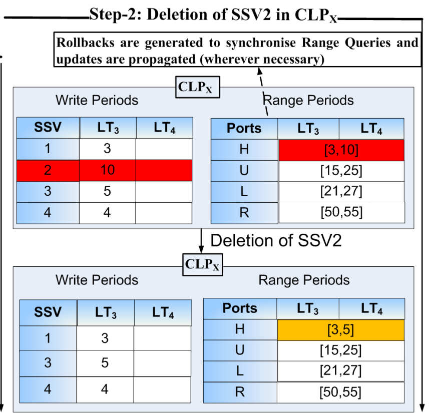
Write Periods

SSV	LT_3	LT_4
1	3	
2	10	
3	5	
4	4	

Range Periods

Ports	LT_3	LT_4
H	[3,10]	
U	[15,25]	
L	[21,27]	
R	[50,55]	

(a) Step-1: GVT Calculation in CLP_X


Step-2: Deletion of SSV2 in CLP_X
Rollbacks are generated to synchronise Range Queries and updates are propagated (wherever necessary)
CLP_X
Write Periods
Range Periods
Deletion of SSV2
CLP_X
Write Periods
Range Periods

Write Periods

SSV	LT_3	LT_4
1	3	
2	10	
3	5	
4	4	

Range Periods

Ports	LT_3	LT_4
H	[3,10]	
U	[15,25]	
L	[21,27]	
R	[50,55]	

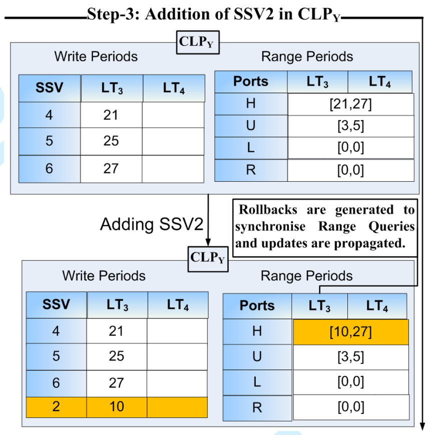
Write Periods

SSV	LT_3	LT_4
1	3	
3	5	
4	4	

Range Periods

Ports	LT_3	LT_4
H	[3,5]	
U	[15,25]	
L	[21,27]	
R	[50,55]	

(b) Step-2: Deleting SSV2 in CLP_X


Step-3: Addition of SSV2 in CLP_Y
CLP_Y
Write Periods
Range Periods
Adding SSV2
Rollbacks are generated to synchronise Range Queries and updates are propagated.
CLP_Y
Write Periods
Range Periods

Write Periods

SSV	LT_3	LT_4
4	21	
5	25	
6	27	

Range Periods

Ports	LT_3	LT_4
H	[21,27]	
U	[3,5]	
L	[0,0]	
R	[0,0]	

Write Periods

SSV	LT_3	LT_4
4	21	
5	25	
6	27	
2	10	

Range Periods

Ports	LT_3	LT_4
H	[10,27]	
U	[3,5]	
L	[0,0]	
R	[0,0]	

(c) Step-3: Adding SSV2 in CLP_Y

Fig. 7: State changes in Write Periods and Range Periods table during the migration of an SSV (SSV_2).

The relationship between the *SSV range* and *RQ size* experimental parameters is determined by the area polled by the Range-Queries and the amount of clustering of the SSVs in the 2-dimensional experimental area. When the area polled by the Range-Queries is small (*RQ size* small), and the SSVs are clustered tightly around the randomly chosen centre-points (*SSV range* small), there is a high probability of the Range-Queries only accessing few if any SSVs everytime they are issued. When the *RQ size* parameter is increased while the *SSV range* remains small, the probability of accessing SSVs increases, and if one SSV is accessed, the probability of accessing the other SSVs in the cluster is also high. Given that the *RQ size* parameter adjusts the radius of the Range-Query, the probability increase is likely to be quadratic, although diminished by overlapping areas. If the *SSV range* parameter is increased while the *RQ size* remain small, the probability of accessing the SSVs increases as well, as they are more spread out over the simulation area but the probability of accessing more SSVs in a cluster decreases proportionally. When both the *SSV range* and *RQ size* parameters are increased in unison a certain average behaviour can be expected. Note that for

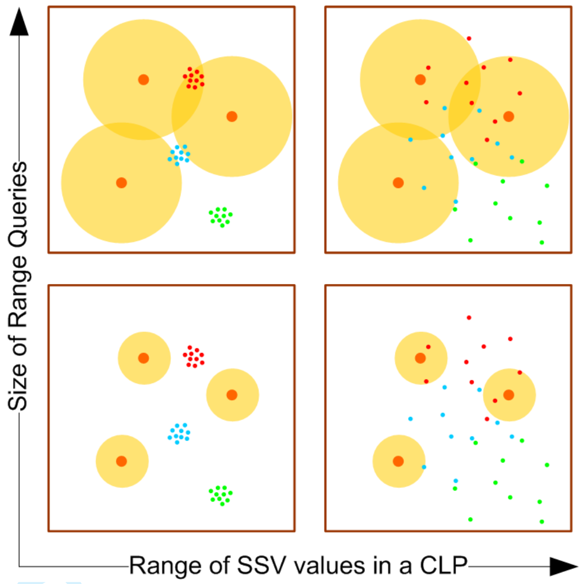


Fig. 8: The logical view of the traces used to investigate the extent of Range-Query propagation under different conditions. The SSVs maintained by each CLP share the same colour.

both parameters a phase transition can be expected when they are increased. For both parameters, each increase of the parameter value will also increase the probability of accessing more SSVs with the rate of increase flattening out and eventually levelling off. The rate of increase will be smaller for the *SSV range* parameter than it is for *RQ size* parameter.

The computational infrastructure used was the Midlands eScience Center (MeSC) cluster environment[1]. Each worker node of the cluster comprises 2 GBytes of memory and 2 Intel Xeon 3GHz processors. All worker nodes are accessible from the clusterś master node connected through a $100MBps$ fast ethernet running Red Hat Enterprise Linux AS release 3.2, kernel version $2.4.21$. For communication, *LAM/MPI (*$7.1.3$*)* [Squyres and Lumsdaine 2003] libraries were utilised.

5.2. Static Tree Analysis

The experiments presented in this section are designed to analyse the performance and interrelationship of three concurrent operations: Range-Query propagation; Synchronisation via Range-Query Rollback; and Range Update propagation in response to write/anti-write patterns.

5.2.1. Range-Query Propagation. This experiment aims to analyse the extent of propagation of a Range-Query (how far a query travels through the tree) as a function of *SSV range* and *RQ size*. The propagation extent of a Range-Query is determined by how quickly the query reaches a port that holds no relevant values beyond it. We therefore hypothesised that, on average, larger queries will propagate further through the CLP graph, and that the larger the range of values CLPs hold, the further the average Range-Query will propagate. We measured propagation extent using the average number of hops a Range-Query performs (since processing occurs in parallel at

[1]`http://www.ep.ph.bham.ac.uk/general/escience-cluster/`

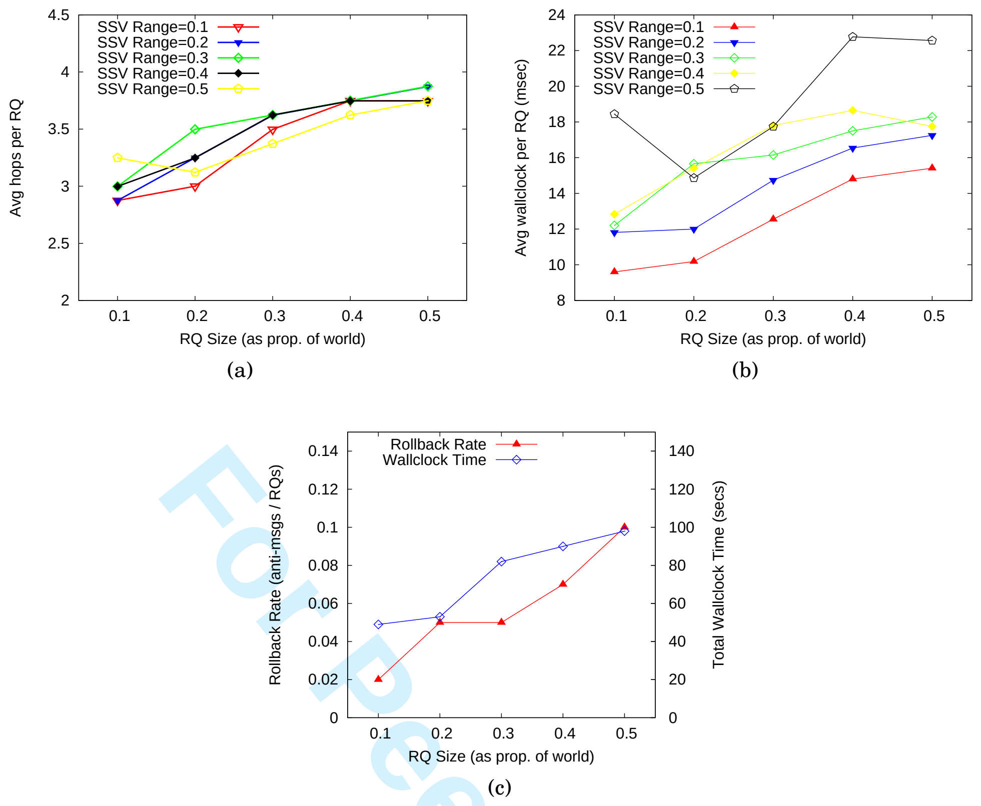


Fig. 9: (a) Average Hops per Range-Query for Different Query Size and SSV Range values (b) Average Walltime per Range-Query for Different Query Size and SSV Range values (c) Relationship Between Range-Query Size and the Incidence of Rollback, and its Effect on Execution Time.

several CLPs, this is taken to be the size of the set of CLPs visited by the query). We also measured the corresponding overall effect on system performance by also recording the average wallclock time between an ALP issuing a Range-Query and obtaining the response.

As figures 9a and 9b both the average hops and wallclock time per query confirm this linear relationship of both Range-Query size and SSV value range to the propagation extent of Range-Queries. The average hops varies from 2.875 (with *RQ size*= 0.1 and *SSV range* = 0.1) to 3.8743 (with *RQ size*= 0.5 and *SSV range* = 0.3), whereas wallclock time per Range-Query ranges from 9.6 msec (with *RQ size*= 0.1 and *SSV range* = 0.1) to 22.77 msec (with *RQ size*= 0.4 and *SSV range* = 0.5).

5.2.2. Range Update Rate. The second experiment analyses the relationship between Write operations and the consequent rate at which Range Updates occur. For this experiment ALPs perform no Range-Queries and simply issue a single Write operation at each logical time step to one of the SSVs in the simulation. The initial distribution of SSVs to CLPs is random and the choice of which SSV to write at each step is also random. To variables parameterise the traces for this experiment:

Write Delta	SSVs/CLP 2	SSVs/CLP 4	SSVs/CLP 8	SSVs/CLP 16	SSVs/CLP 32
0	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.2655	0.1422	0.0536	0.0495	0.0253
2	0.4023	0.2226	0.0998	0.0736	0.0374
4	0.4690	0.2539	0.1170	0.0854	0.0435
8	0.5165	0.2924	0.1293	0.0916	0.0503

Table I: The Range Update Rate for Different Combinations of Write Delta and SSVs per CLP.

— *SSV per CLP*: the number of variables stored by each CLP and
— *Write Delta*: when a write is performed the ALP must choose a new value. The write delta is the metric distance between the SSVs current value and the new value that is written.

Range Updates by a CLP whenever the range covered by the local set of SSVs changes. Accordingly, the hypothesis for this experiment is that Range Updates will be generated more frequently when the SSV set is smaller and when consecutive writes change the value of an SSV by a larger amount. The actual metric used to quantify the frequency of Range Updates is ratio of Range Updates to Write Operations, hereafter termed the 'Range Update Rate'.

The results are presented in table I. As predicted, both parameters are strongly correlated to the Range Update Rate. In the extreme case of a 0-value Write Delta, no Range Updates occur, since the Ranges covered by CLPs never change. As Write Delta rises, each write has a higher chance of causing a Range Update. In the opposite extreme of Write Delta $= 8$ and SSVs per CLP $= 2$, *every* write will cause the local set of the host CLP to change[2]. The rate varies from 0 (with *SSV per CLP*= 2 and *Write Delta* = 0) to 0.5 (with *SSV per CLP*= 2 and *Write Delta* = 8). One can notice as *SSV per CLP* increases, the rate decreases linearly as there is a lower chance to send range updates.

5.2.3. Rollback Rate. The final experiment of this set considers a more complete trace which contains both Range-Query and Write operations, creating the potential for Rollback. For this experiment both the Write Delta Size and the SSVs per CLP were set statically at the values 1 and 8 respectively and the Range-Query size varied between 0.1 and 0.5 as a proportion of the size of the simulated world. In these conditions the Range Updates which occur will remain relatively static between consecutive runs. At the same time, the extent of Range-Query propagation is expected to increase in proportion to its size, as will the average wallclock time required to complete them (as observed in Section 5.2.1 above). Correspondingly, we predict Range-Queries with a large propagation extent through the tree to be more likely to be rolled back and, consequently, increase the overall wallclock time of the simulation. For a given simulation execution the actual probability of a Range-Query being rolled back can be estimated as the ratio of Anti-Messages to Range-Queries, hereafter termed the Rollback Rate. In the absence of other variables we expect this property to be proportional to wallclock time.

The correlation between Query Size and Rollback Rate and the corresponding impact on total Execution Time is shown in figure 9c and strongly agrees with the predictions given above.

[2]Note that the local set changing will not necessarily cause the propagation of a Range Update message to a neighbouring CLP, see section 3.2.

5.3. Dynamic Tree Analysis

The overall objective of state migration is to improve the performance of the system by moving SSVs closer to the accessing ALPs, thus reducing the number of hops needed to access the SSVs resulting in a reduction in the average access cost of the SSVs. But state migration also comes with a cost; as described in section 4, each state migration initiated also initiates a rollback and moving SSVs means an increase in the number of updates when they are moved from one CLP to another. In this section we present an experimental investigation into both effects in order to evaluate whether the access cost reduction of state migration outweights its inherent cost. The aim of this investigation is twofold: (a) analyse the impact of Range-Query propagation and State Migration on the access cost of the system; and (b) analyse the impact of increased rollback volatility and State Migration cost on average simulation time.

Experimental results both with and without state migration are presented to allow for a clear comparison between the two configurations.

5.3.1. Range-Query Propagation. In this experiment we want to quantify the effect Range-Query propagation and state migration has on the access cost of the SSVs. To isolate this effect, the traces generated for this experiment did not include any write-events. As such, any rollbacks initiated will be caused by state migration alone without further dilution from non-deterministically caused rollbacks by write-events in the traces themselves. With the simulations run for 1000 ticks, the ALPs will issue 1000 range queries as well.

Range-Query propagation is measured by the number of hops it takes to fetch the SSVs in the CLP tree. Without state migration the number of hops required would be 7; 3 hops from the leaf CLP to the root, 3 return hops, and an extra hop to the ALP. The same hop counting method is used for both experiments. The number of SSVs accesses, and thus the total number of hops for an experiment, is determined by the combination of the experimental parameters: *SSV range* and *RQ size*.

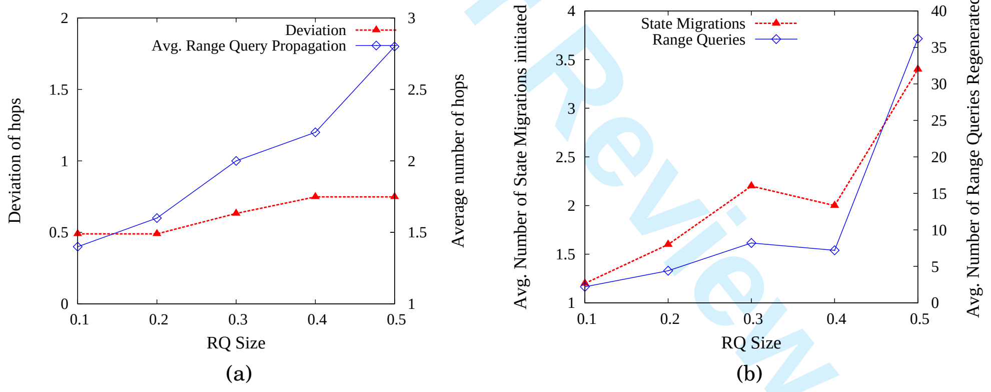


Fig. 10: Average number of hops (with its deviation from the mean) and average number of State Migrations initiated and Range-Queries regenerated for varying *RQ size*.

Figure 10a shows the average number of hops required to access SSVs and their deviation from the mean for varying *RQ size* values. The results were averaged over the various *SSV range* values without loss of accuracy. The results suggest a linear relationship between the average number of hops and the *RQ size* parameter, with the average number of hops ranging from 1.4 to almost 3 for *RQ size* from 0.1 to 0.5. In this range, the standard deviation from the average number of hops indicates a linear increase from 0.4 to 0.7. The results suggest that the increase in variance is caused by a higher probability for SSVs to be localised with minimum or no overlap with range queries issued from different ALPs for smaller *RQ size* values. With minimum or no overlap, SSVs will eventually be migrated to the leaf CLPs. When the *SSV range* and/or *RQ size* parameter is increased, a correspondingly higher likelihood of having Range-Query overlapping each other means an increased likelihood of SSVs remaining at intermediate CLPs so as to be more efficiently available for several ALPs. This increases the spread of the average number of hops required to access these SSVs with the corresponding increase in the variance and standard deviation of this measure. Overall, the experimental results show a clear reduction of the average number of hops required to access the SSVs from the constant 7 hops required to access them in the root CLP. With an average of around 3 hops required in the worst case, this shows that state migration has a substantial effect on Range-Query propagation.

Range-Query propagation on its own only shows one aspect of overall picture though. State Migration's extra cost has to be taken into account. We express the extra cost incurred by state migration by measuring the number of state migrations initiated and the number of Range-Queries regenerated with the underlying assumption that the number of State Migrations initiated is directly correlated with the number of range queries regenerated.

Figure 10b shows the average number of state migrations initiated and Range-Queries regenerated for varying *RQ size* values. The figure supports our expectation that the average number of State Migrations initiated increases with increased *RQ size* values. The average number of Range-Queries regenerated follows this trend closely with the maximum number of state migrations initiated reaching almost 4 and the number of Range-Queries regenerated almost 35 for *RQ size* 0.5. Although the number of Range-Queries regenerated is almost a factor of 8 times the number of state migrations initiated, compared to the total number Range-Queries generated (1000 per ALP), this should still be considered to a low number.

Thusfar, the results show that including state migration decreases the number of hops, but also increases the number of Range-Queries regenerated, in turn increasing the number of accesses. What remains is to see how this contributes to the overall access cost of the simulation. The access cost of a simulation is the sum of the cost of accessing SSVs in the CLP tree. The cost of accessing a SSV in the CLP in turn is subject to the number of Range-Queries (the number of accesses) and the number of hops needed to reach them in the CLP tree.

The access cost of the simulation without and with state migration is compared by calculating the difference ratio in percentages called the *Cost Reduction*: C_r. *Cost Reduction* (C_r) is calculated as follows:

$$C_r = \frac{C_{-SM} - C_{+SM}}{C_{-SM}} \cdot 100 \qquad (1)$$

With C_{-SM} the access cost of the simulation without state migration and C_{+SM} the total access cost of the simulation with state migration.

Figure 11a shows the *Cost Reduction* for different *SSV range* and *RQ size* values. The C_r with state migration averaged over both *SSV range* and *RQ size* is 47.12 but the figure suggests a lot of variance for different *SSV range* and *RQ size* combinations with a decrease in C_r when *SSV range* and *RQ size* increase. The highest *Cost Reduction* was achieved with *SSV range* 0.3 and *RQ size* 0.1 for a reduction of almost 70%, while the lowest *Cost Reduction* was found to be with *SSV range* 0.5 and *RQ size* 0.5.

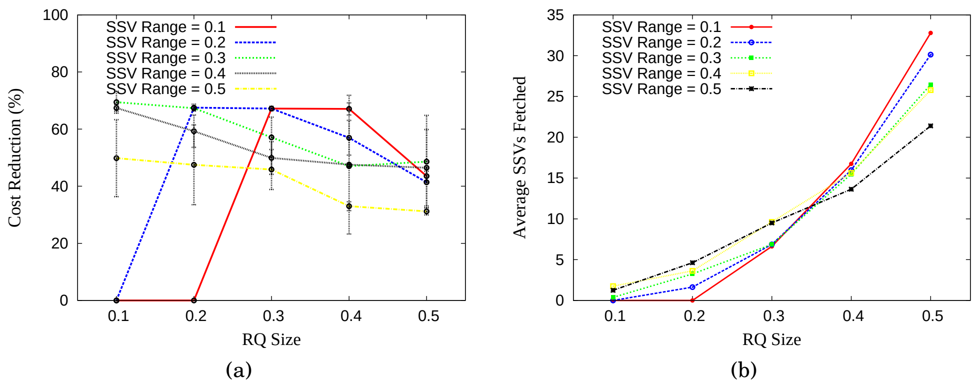


Fig. 11: (a) Cost reduction C_r in percentages for different *SSV range* and *RQ size* combinations with their standard deviations (b) Average SSVs fetched by a Range-Query for different *SSV range* and *RQ size* combinations

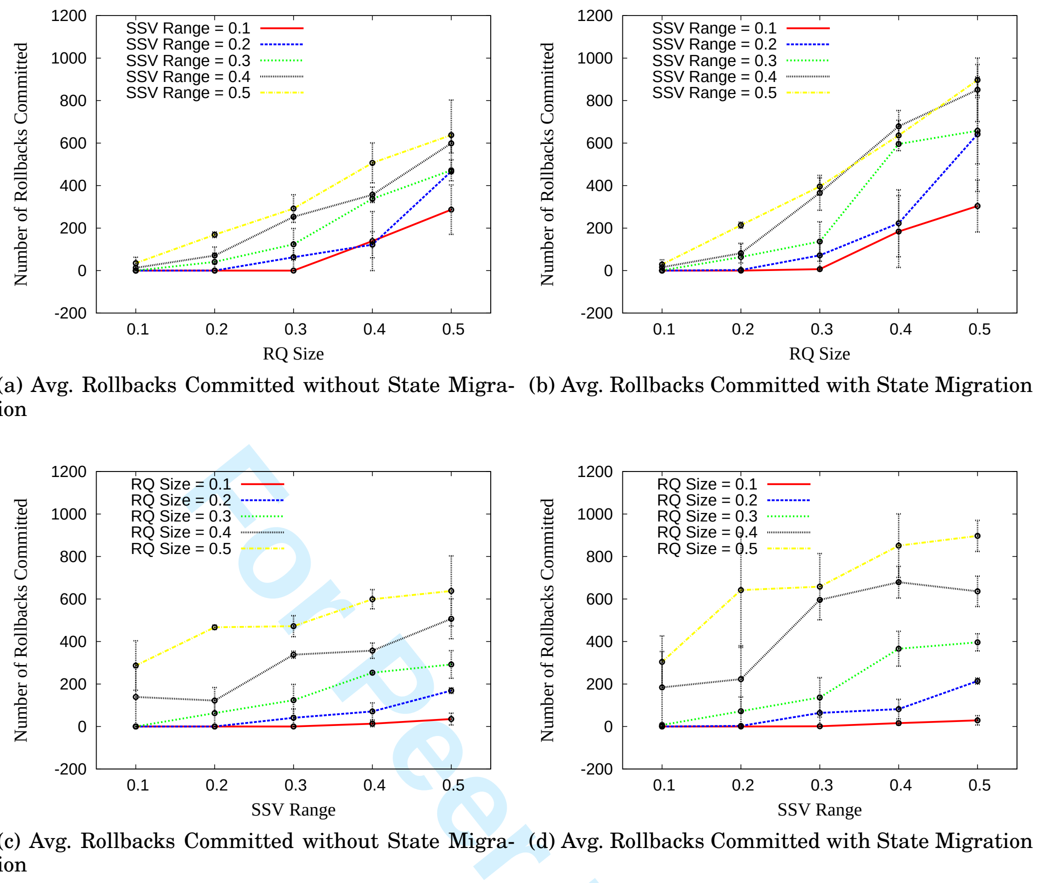
Figure 11a also shows the standard deviations from the averages *Cost Reduction*. The trend shown in the figure suggests that the variance decreases when the *SSV range* and *RQ size* values increase.

For a few *SSV range RQ size* combinations, no observable *Cost Reduction* was observed, meaning that the access cost both without and with state migration remained the same. To investigate this we present the average number of SSVs fetched by the Range-Queries for different *SSV range* and *RQ size* combination in figure 11b.

The figure shows that when the *SSV range* and *RQ size* parameters are small, on average no SSVs were fetched. For these *SSV range* and *RQ size* values, the area covered by the range queries is so small, or, alternatively, the area in which the SSVs are concentrated is so small, that no SSVs matched the Range-Queries issued by the ALPs. Since state migration only updates access cost with SSVs that match the range query predicate, in these instances, no state migrations were ever initiated, and without these no cost reduction was affected. In conclusion, the results show that for state migration to effect the greatest cost reduction, the *SSV range* and *RQ size* parameters should be large enough to at least fetch some SSVs while keeping them small enough so that SSVs access is localised enough for SSVs to migrate closest to the ALPs that will access them. In these circumstances, state migration will reduce Range-Query propagation significantly with a enough positive effect on the access cost of the simulation to off-set the extra cost incurred by state migration.

5.3.2. Rollback volatility. Thusfar we have considered experimental traces without any write-events in order to clearly expose the effect state migration has on access cost. Experimental traces without write-events do no initiate any rollbacks from straggler writes, and do not expose the effects of the interactions between rollbacks initiated from different sources. In the following experiment we do include write-events in the experimental traces so that we can fully assess the impact of rollbacks on overall simulation time. The same experimental setup as used in the first experiment is used here with write-events increasing the value of the SSVs by 1. Write-events are issued by all ALPs to SSVs randomly selected from all SSVs. As before, all SSVs are placed at the root CLP in the CLP tree, with the *SSV range* and *RQ size* parameters varied.

Figure 12 shows the rollbacks committed per ALP for varying *SSV range* and *RQ size* combinations. Four graphs are shown with the first two graphs show the results when



(a) Avg. Rollbacks Committed without State Migration

(b) Avg. Rollbacks Committed with State Migration

(c) Avg. Rollbacks Committed without State Migration

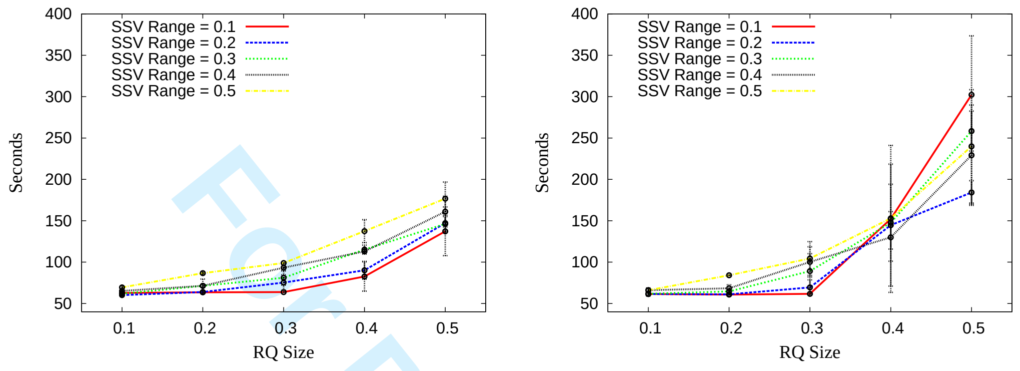
(d) Avg. Rollbacks Committed with State Migration

Fig. 12: Rollbacks committed per ALP for varying *RQ size* and *SSV range* with their standard deviations.

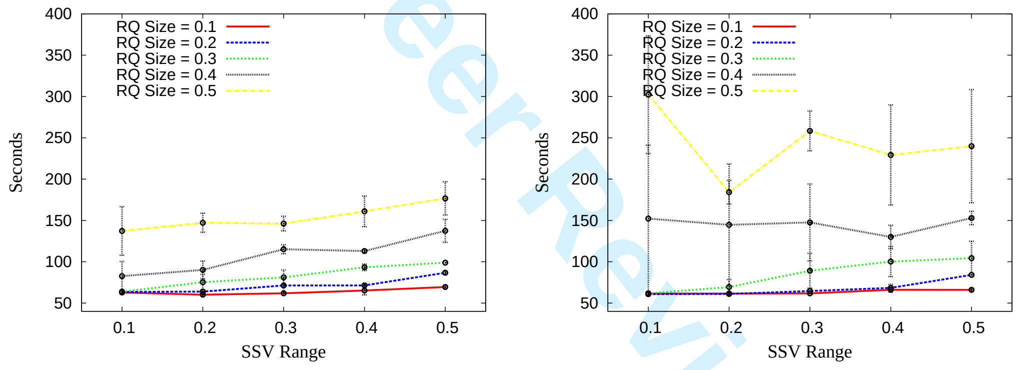
the *RQ size* parameter was varied plotted for different *SSV range* parameter values (with and Without SM). The last two graphs show the results with the *SSV range* parameter was varied plotted for different *RQ size* parameter values (with and Without SM). As the results were averaged over 3 runs with different pseudo random number generator seeds, the standard deviation for each result in the graphs is depicted using error-bars.

The order in which the events, both reads and writes, are processed by PDES-MAS is important in that an ALP commits a rollback if it arrives with a timestamp earlier that its local time (straggler-events), committing the event otherwise [Logan and Theodoropoulos 2001; Lees et al. 2008]. A committed rollback also has the potential to regenerate Range-Queries. As such, we expect an increase in the number of rollbacks committed by the ALPs for increasing *SSV range* and *RQ size* parameter values. In addition, with state migration moving SSVs around the CLP tree, we also expect more rollbacks regenerating Range-Queries.

The result presented in figure 12 bear out this expectation. The number of rollbacks increase almost linearly with larger *SSV range* and *RQ size* parameter values. With *RQ size* 0.1, the number of committed rollbacks is close to 0, both with and without state migration. Without state migration, the maximum number of rollbacks reaches 800, with state migration it reaches 1000, for both *SSV range* and *RQ size* 0.5. Although there is substantial variance in the standard deviation values, the overall trend is that it is relatively small upto *SSV range* and *RQ size* 0.3 suggesting few Range-Queries regenerated. Beyond that there is a large amount of Range-Query regeneration. Beyond that the deviation increases to 170 (without state migration) and 250 (with state migration), suggesting a large amount of Range-Query regeneration. In general, as expected, the number of rollbacks committed with State Migration is higher than with state migration.



(a) Average Simulation Time without State Migration (b) Average Simulation Time with State Migration



(c) Average Simulation Time without State Migration (d) Average Simulation Time with State Migration

Fig. 13: Average Simulation Time for varying *RQ size* and *SSV range* with their standard deviations.

Based on these results we expect that as the *SSV range* and *RQ size* parameters increase, the simulation time required to finish the experimental traces will also increase. Figure 13 show the average simulation time for varying *SSV range* and *RQ size* parameter values with their standard deviations. The layout of the graphs is the same as described above for the number of rollbacks committed.

Figure 13 suggests that the average simulation time increases with increases in both the *SSV range* and *RQ size* parameters. Without state migration the average simulation time ranges from 60 seconds for *SSV range* 0.2 and *RQ size* 0.2 to 176 seconds for

SSV range and *RQ size* 0.5. Again standard deviations over the averaged simulation time varies but suggest a minimum for *SSV range* and *RQ size* 0.3 beyond which the standard deviation reaches 20. This correlates strongly with the trend shown for the number of rollbacks committed as presented in figure 12.

With state migration the average simulation time increases linearly upto *RQ size* 0.3, increasing exponentially beyond that. The average simulation time ranges from 60 seconds for *SSV range* 0.2 and *RQ size* 0.1 to reach 302 seconds for *SSV range* 0.1 and *RQ size* 0.5. Comparing the average simulation time without state migration with the average simulation time with state migration we see a small reduction of the average simulation time until *SSV range* and *RQ size* reach 0.3, after which the average simulation time with state migration exceeds that of the average simulation time without state migration.

This does not correlate exactly with the trend shown in figure 12. With the query propagation trend with state migration as shown in figure 10a we would expect more localised access patterns to effect a reduction of the number of rollbacks and consequently the simulation time. What should be noted though is that the writes in the simulation time are issued to random SSVs in the CLP tree without consideration of where these SSVs are localised in relation to the Range-Queries. This has two effects on the simulation based on the location of the SSV in the CLP tree.

Firstly, as the *RQ size* and *SSV range* increase, the overlaps of the range queries between different ALPS also increase. Though query propagation is reduced significantly by state migration (see figure 10a), writes to random SSVs in this scenario also increase the overlaps between ALPs. The reasoning behind this is that as the random writes change the value of a SSV, the possibility of the SSV overlapping Range-Queries of different ALPS also increases as the *SSV range* and *RQ size* parameters increase. To further quantify this effect we measured the *Query Response Time* over a simulation run. With state migration, changes in the *Query Response Time* should be minimal but with state migration the *Query Response Time* depends on Range-Query propagation in the CLP tree, and variances could be large.

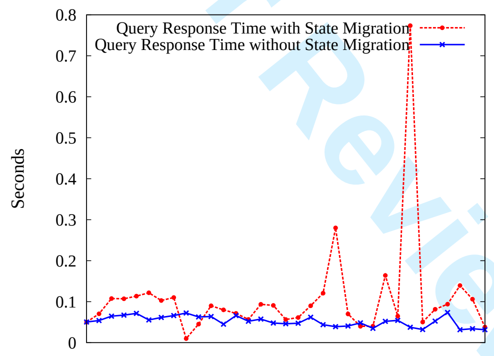


Fig. 14: Figure depicts Query Response Time with and without State Migration for a typical simulation run using random traces.

Figure 14 illustrates this pattern by showing *Query Response Time* during a typical simulation run both with and without state migration. As expected, figure 14 shows that *Query Response Time* for simulation without state migration hardly alters during the run. *Query Response Time* with state migration however shows an initial increase but a subsequent reduction, suggesting more localised access patterns. But as the simulation progresses we see peeks of dramatic increases in *Query Response Time* followed by equally dramatic decreases. This suggests an interaction between the random writes and the State Migration with a dramatic decrease in efficiency by a confluence of random writes followed by state migration reacting on this. State migration is always reacting on these instances and in the end, averaged over the run *Query Response Time* is negatively affected.

Secondly, Rollback depth is increased with localised access patterns. Rollback depth is measured as the difference between the local time of a rollback and the ALP committing the rollback. All Range-Queries and writes generated in this time period will be rolled back and regenerated. When *RQ size* is small, the SSVs are moved much closer to the ALPs accessing them (see figure 10a) and as such we should observe a significant reduction in simulation time. However, as the SSVs move closer to the leaf CLPs, a random write from a remote ALP will need more hops to update the SSV while having a higher probability of triggering a rollback because of the increased time needed to traverse the CLP tree as well as increasing the *Rollback Depth*.

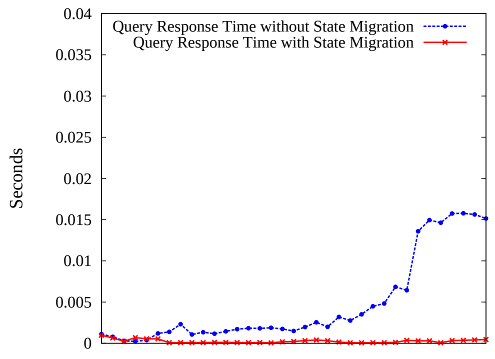
RQ size	***SSV range***				
	0.1	0.2	0.3	0.4	0.5
0.1	0	0	14.5	19.52	15.91
0.2	0	13.85	1.91	12.23	6.10
0.3	3.42	9.09	9.24	4.68	4.16
0.4	7.19	8.57	4.35	3.34	2.94
0.5	9.93	2.48	2.46	3.34	3.37

Table II: Average Rollback Depth for different *SSV range* and *RQ size* combinations

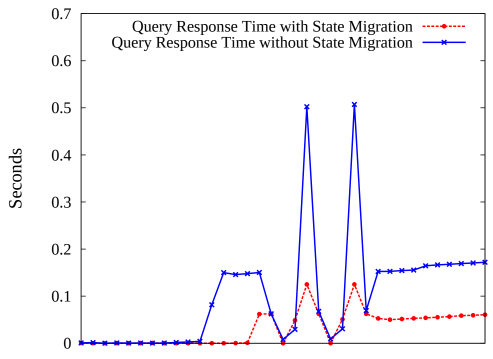
This effect is illustrated by the average rollback depths found for different *SSV range* and *RQ size* parameter values as shown in table II. Although no linear trend can be distinguished for all *SSV range* and *RQ size* combinations, we note that for *RQ size* 0.1 this depth is almost 10 whereas for *RQ size* 0.5 it is just 4.3. This suggests that as the *RQ size* increases, the rollback depth decreases with more SSVs remaining at intermediate nodes where rollbacks are reached faster. Overall this suggests that although the number of rollbacks committed are fewer (see figure 12), the depth of these rollbacks is higher, making them more expensive to perform and thus negatively affecting simulation time.

The scenario of random writes investigated so far represents the worse case, wherein lack of locality results to poor SSV distribution in the tree. The philosophy behind PDES-MAS is the exploitation of locality expected in agent-based models. Figure 15 presents Query response times from a well known agent-based model exhibiting locality, namely Boids [Reynolds 1987]. Boids is widely used to simulate flocking behaviour (of birds or herds of animals). Agents (boids) are collaborative in nature, flock together and share common interest rather than individual planning behaviour. Figure 15 depicts query response times for a typical Boids simulation run with two different experiment parameters: *RQ size* and *Velocity*, the speed in which a Boid moves.

As boids move towards each other they tend to form initially small clusters. As the simulation progresses in time, these small clusters merge together in a smaller number of larger clusters. When *RQ size* and *Velocity* values are small, the cluster formation is slow and predictable, as illustrated in figure 15 where more subtle variations in response times are observed due to a more even distribution of SSVs in the tree and smaller rollback costs.



(a) Query Response Time with experimental parameters *RQ size* = 0.1 and *Velocity* = 2



(b) Query Response Time with experimental parameters *RQ size* = 0.5 and *Velocity* = 10

Fig. 15: Query Response Time for a typical simulation run using Boids.

As these parameters increase, the formation is fast and more volatile, resulting to more centralised placement of SSVs in the CLP tree increasing CLP loads and rollback costs, as illustrated by more violent fluctuations in (figure 15b). In all these cases, query response times are clearly reduced with the introduction of state migration.

6. CONCLUSIONS

Range-Queries are key operations in MAS models, representing the spatial perceptive abilities of the agents in the MAS. Their implementation in a distributed environment has been acknowledged as a challenging endeavour while their efficient realisation is crucial for the performance of the distributed simulation system.

This paper has presented a detailed design for logical-time synchronised instantaneous Range-Queries in the context of PDES-MAS, a framework for the distributed simulation of MAS models. This design is based on a paradigm of routing Queries around a distributed data structure by matching the Query's range predicate against explicitly maintained information about state variables that lie beyond a given edge in the graph (Range-Based routing). The effectiveness of the proposed algorithms have been evaluated under different conditions.

Our analysis has indicated that the paradigm of handling synchronised Range-Queries via Range-Based routing ties performance closely to Query and Update patterns. The more dynamic write patterns are - and the more violently these can change the ranges covered by CLPs - the more frequently Range Updates will occur. Concurrently, the larger a proportion of the world a given Range-Query covers, the more likely a given Query will be at risk of rollback due to straggler updates in the data structure. The introduction of state migration reduces the propagation extent of the Queries but in principle, introduces extra overhead in the system. Our analysis has shown however that the extra cost of migration process does not necessarily compromise the performance of the system. A significant reduction in the access latency and cost of the simulation for varying Range-Query Sizes and SSV Range Values has been demonstrated. The volatility of rollbacks has been shown to have a direct relation with varying Range-Query sizes and SSV Range values and an adverse effect with State Migration with minimum gain.

As part of the *MWGrid*[3] project, PDES-MAS is being utilised to support large scale agent-based simulations designed to investigate medieval military logistics. Future work will evaluate the proposed Range-Query algorithms in the context of MWGrid simulations.

MAS simulations are increasingly recognised as a key paradigm for deep Big Data Analytics [Wallis 2011; FuturICT-Proposal]. As we are moving to the exascale computing era, there is a pressing need to support exteme-scale MAS simulations which will be accessing exascale historical and streaming data [Macal and North 2008; Kennedy et al. 2011]. Efficient realisation of synchronised Range-Queries will be the key factor to meet the performance and scalability requirements of this new generation of data-intensive analytics approaches. The work presented in this paper aspires to contribute to this challenging endeavour.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

BARRUS, J. W., WATERS, R. C., AND ANDERSON, D. B. 1996. Locales: Supporting large multiuser virtual environments. *IEEE Computer Graphics and Applications 16*, 50–57.

BERRACHED, A., BEHESHTI, M., SIRISAENGTAKSIN, O., AND DEKORVIN, A. 1998. Approaches to multicast group allocation in hla data distribution management. In *In Proceedings of the 1998 Spring Simulation Interoperability Workshop*.

BHARAMBE, A. R., AGRAWAL, M., AND SESHAN, S. 2004. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Comput. Commun. Rev. 34,* 4, 353–366.

BOUKERCHE, A. AND ROY, A. 2002. Dynamic grid-based approach to data distribution management. *J. Parallel Distrib. Comput. 62,* 3, 366–392.

CRAENEN, B., SURYANARAYANAN, V., AND THEODOROPOULOS, G. March, 2011. A middleware for interfacing with simulation systems of multi-agent models. *2nd Workshop on Distributed Simulation & Online gaming (DISIO 2011)*. Barcelona, Spain.

CRAENEN, B., THEODOROPOULOS, G., SURYANARAYANAN, V., GAFFNEY, V., MURGATROYD, P., AND HALDON, J. March 21, 2010. 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, 6:1–6:8.

EGGERS, S. J. AND KATZ, R. H. 1989. Evaluating the performance of four snooping cache coherency protocols. *SIGARCH Comput. Archit. News 17,* 3, 2–15.

EWALD, R., DAN, C., OGUARA, T., THEODOROPOULOS, G., LEES, M., LOGAN, B., OGUARA, T., AND UHRMACHER, A. M. 2006. Performance analysis of shared data access algorithms for distributed simulation of MAS. In *Proceedings of the 20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006)*, S. J. Turner and J. Lüthi, Eds. IEEE Press, Singapore, 29–36.

FUTURICT-PROPOSAL. Global earth simulator. `http://www.futurict.eu/`.

GELERNTER, D. 1985. Generative communication in linda. *ACM Trans. Program. Lang. Syst. 7,* 1, 80–112.

HOOK, D. V., RAK, S. J., AND CALVIN, J. O. 1994. Approaches to relevance filtering. In *In Eleventh Workshop on Standards for the Interoperability of Distributed Simulations*. 26–30.

HU, S.-Y. AND LIAO, G.-M. 2004. Scalable peer-to-peer networked virtual environment. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, New York, NY, USA, 129–133.

KENNEDY, C., THEODOROPOULOS, G., SORGE, V., FERRARI, E., LEE, P., AND SKELCHER, C. November 07, 2011. Data driven simulation to support model building in the social sciences. *Journal of Algorithms & Computational Technology (JACT) Volume 5/Number 4*, 561–582. Special Issue on DDAS.

KNUTSSON, B., GAMES, M. M., LU, H., XU, W., AND HOPKINS, B. 2004. Peer-to-peer support for massively multiplayer games. In *In Proceedings of INFOCOMM 2004*.

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

LEES, M., LOGAN, B., AND THEODOROPOULOS, G. 2003. Adaptive optimistic synchronisation for multi-agent simulation. In *Proceedings of the 17th European Simulation Multiconference (ESM 2003)*, D. Al-Dabass, Ed. Society for Modelling and Simulation International and Arbeitsgemeinschaft Simulation, Society for Modelling and Simulation International, Delft, 77–82.

LEES, M., LOGAN, B., AND THEODOROPOULOS, G. 2008. Using access patterns to analyze the performance of optimistic synchronization algorithms in simulations of mas. *Simulation 84,* 10/11 (October), 481–492.

LEES, M., LOGAN, B., AND THEODOROPOULOS, G. 2009. Analysing probabilistically constrained optimism. *Concurrency and Computation: Practice and Experience 21,* 11 (August), 1467–1482.

LOGAN, B. AND THEODOROPOULOS, G. 2001. The distributed simulation of multi-agent systems. *Proceedings of the IEEE 89,* 2 (Feb), 174–186.

MACAL, C. M. AND NORTH, M. J. 2008. Agent-based modeling and simulation for exascale computing. *SciDac Review*, 34–41. http://www.scidacreview.org/0802/html/abms.html.

MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BRUTZMAN, D. P., AND BARHAM, P. T. 1995. Exploiting reality with multicast groups. *IEEE Comput. Graph. Appl. 15,* 5, 38–45.

MATTERN, F. 1993. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel Distributed Computing 18,* 4, 423–434.

MEHL, H. AND HAMMES, S. 1993. Shared variables in distributed simulation. In *PADS '93: Proceedings of the seventh workshop on Parallel and distributed simulation*. ACM, New York, NY, USA, 68–75.

MINSON, R. AND THEODOROPOULOS, G. 2007. Adaptive support of range queries via push-pull algorithms. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, Washington, DC, USA, 53–60.

MORGAN, G., STOREY, K., AND LU, F. 2004. Expanding spheres: A collision detection algorithm for interest management in networked games. In *In Proceedings of the Entertainment Computing ICEC 2004: Third International Conference*. 435.

MORSE, K. L. 2000. An adaptive, distributed algorithm for interest management. Ph.D. thesis, Information and Computer Science. Chair-Bic, Lubomir and Chair-Dillencourt, Michael.

MORSE, K. L. AND ZYDA, M. 2002. Multicast grouping for data distribution management. *Simul. Pr. Theory 9,* 3-5, 121–141.

OGUARA, T., CHEN, D., THEODOROPOULOS, G., LOGAN, B., AND LEES, M. 2005. An adaptive load management mechanism for distributed simulation of multi-agent systems. In *Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, A. Boukerche, S. J. Turner, D. Roberts, and G. Theodoropoulos, Eds. IEEE Press, Montreal, Quebec, Canada, 179–186.

PAGEL, B.-U., SIX, H.-W., TOBEN, H., AND WIDMAYER, P. 1993. Towards an analysis of range query performance in spatial data structures. In *PODS '93: Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, New York, NY, USA, 214–221.

PAGIAMTZIS, K. AND SHEIKHOLESLAMI, A. 2006. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits 41,* 3 (March), 712–727.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '87. ACM, New York, NY, USA, 25–34.

SQUYRES, J. M. AND LUMSDAINE, A. 2003. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*. Number 2840 in Lecture Notes in Computer Science. Springer-Verlag, Venice, Italy, 379–387.

SURYANARAYANAN, V., CRAENEN, B. G. W., AND THEODOROPOULOS, G. K. 2010. Synchronised range queries in distributed simulations of multi-agent systems. In *Proceedings of the 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications*. DS-RT '10. IEEE Computer Society, Washington, DC, USA, 79–86.

SURYANARAYANAN, V., MINSON, R., AND THEODOROPOULUS, G. K. 2009. 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, 41–47.

WALLIS, L. March 01, 2011. Big data, analytics, and storytelling. *Causalities: Applied Systems Science, Dynamics and Simulation*. `http://blog.cause-alities.com`.

YU, A. P. AND VUONG, S. T. 2005. Mopar: a mobile peer-to-peer overlay rrchitecture for interest management of massively multiplayer online games. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. ACM, New York, NY, USA, 99–104.

Online Appendix to: Synchronised Range-Queries in Distributed Simulations of Multi-Agent Systems

VINOTH SURYANARAYANAN, University of Birmingham, UK
GEORGIOS THEODOROPOULOS, IBM Research, Ireland

The algorithms implemented in the PDES-MAS system to support Range-Queries are outlined in this appendix. There are four algorithms presented in this paper:

— Algorithm 1 is used to maintain the RangePeriodList data structure in the case of Range Update operations
— Algorithm 2 is used to propagate synchronised range updates through ports
— Algorithm 3 is employed to rollback a Range-Query on receiving a straggler write
— Algorithm 4 is utilised to migrate SSVs across CLPs.

The rest of the appendix describes these algorithms in more detail.

ALGORITHM 1: The algorithm for correct maintenance of a RangePeriodList data structure over a set of SSVs

```
Input: The logical time at which Range Period List is modified.
Result: The Range Period list at time is updated.

Function : update_ range_ list(time);
rp_ split = find_ rp(time) /* Find range period split by the write/anti-write */;
if rp_ split != NULL then
    new_ range = get_ range(ssv_set, time) /* Calculate the range now covered by the SSVs at
    this time */;
    if new_ range != old_ range then
        insert_ rp(rp_split + 1, time, new_range) /* Insert a new RangePeriod for this time */;
        /* For all RangePeriods > time, recalculate their ranges */ for rp in range_
        periods(time) do
            if rp.range != get_ range(ssv_set, rp.time) then
                update_ rp(rp, get_range(ssv_set, rp.time));
            end
        end
    end
end
else
    insert_ rp(start_ of_ list, time, new_ range);
end
```

© YYYY ACM 1539-9087/YYYY/01-ARTA $10.00
DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

ALGORITHM 2: The algorithm to propagate synchronised range updates

Input: A List of Range Updates with originating *port*
Result: The Range Updates are disseminated to other ports.

```
Function : send_ range_ update($range\_update\_list, origin\_port$);
for $ru$ in $range\_update\_list$ do
    /* Iterate through the list of range updates.Each range update has a range associated with a
    time period.*/;
    for $port$ in $CLP.rem\_ports$ do
        old_ range = find_ range($port, ru.time$);
        new_ range = calculate_ range($origin\_port, ru.range, CLP.ports$) /* Recalculate the new
        range using the logic mentioned above. */;
        if old_ range != new_ range then
            send_ range_ update($port, time, new\_range$);
        end
    end
end
```

ALGORITHM 3: The algorithm for calculating whether to rollback a Range-Query in response to a Write Period update. The algorithm for a Range Period update is identical, the only exception being the intersection test is between two ranges rather than a range and a single value.

Input: The logical time at which new value is updated.
Result: Rollbacks may be generated in response to Write Period Update.

```
Function : update_ ssv($time, value$);
old_ wp = find_ wp(time) /* Find the write period split by this write */;
if old_ wp != NULL then
    new_ wp = insert_ wp($old\_wp, time, value$);
    /* for all RQs > time, evaluate rollback conditions */;
    for RQ in old_ wp.queries do
        if $RQ.time > time$ then
            old_ wp.remove(RQ);
            /* Either rollback query or place in new write period */;
            if !RQ.contains($old\_wp.value$) & !RQ.contains($value$) then
                new_ wp.add(RQ)
            end
            else
                rollback(RQ)
            end
        end
    end
end
else
    insert_ wp($start\_of\_list, time, value$)
end
```

ALGORITHM 4: SSV Migration algorithm in PDES-MAS.

```
Input: SSV migration is initiated at $time$.
Result: Selected SSVs are migrated with their entire history.

Function : SelectSSVForMigration($time$);
for all ssvs do
    for all directions do
        /* Get access cost and port cost for each SSV at each port */ if ssv.portcost > PORT.THRESHOLD then
            if ssv.portcost > cost.remainingports then
                MigrateList.add($ssv$);
            end
        end
    end
end
Function : MigrateStateVariables($time$);
/* Iterate through migration state variables */;
for each $dir$ in directions do
    /* ssvs selected for migration at each port */ for $ssvs$ in MigrateList do
        MigrateSSVfromCLP ($ssv$) /* delete ssv from the CLP and updates its new port */;
        DeleteWritePeriods ($ssv$) /* delete write periods of the migrating ssv */;
        UpdateRangePeriods ($ssv$) /* update range periods accordingly */;
        SendRangeUpdates ($time$) /* send range updates accordingly */;
    end
end
```