SYNCHRONISED RANGE QUERIES

by

VINOTH SURYANARAYANAN

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
The University of Birmingham
December 2011

Abstract

Computer simulations have been used more than ever before to embark on developing and understanding complex systems such as Multi-Agent Systems (MAS). MAS are complex, non-deterministic, data-centric behaviour and nature. Simulations play a key role for the designer of an agent based system to experiment and study the impact of different architectures, environment and agent behaviour. As MAS are increasingly used to solve larger and more complex problems, scalability becomes an important issue for the successful deployment. An emerging viable solution is adopting distributed simulation techniques in executing MAS models in parallel. One approach is to distribute the shared state (or environment) of a simulation model across available computing resources. Based on this approach, *PDES-MAS (Parallel and Discrete Event Simulations for Multi-Agent System)* framework is designed to execute large scale models such as MAS. It adopts PDES techniques to distribute and run parallel simulation of Multi-Agent Systems (MAS). Several challenges arise on executing MAS models on a distributed environment, of which one issue that requires focus is data access. Accessing data efficiently in a latency-sensitive and large scale network overlay is a vital requirement for the scalability of the system. Following PDES paradigm, it is also very important that events (accessing shared state) in a parallel and discrete event simulation system are processed in a non decreasing (logical) time stamp order. So, this thesis presents a notion of logical time synchronised range queries to access data and in particular within the PDES-MAS framework. To localise data access, this thesis also provides mechanisms to distribute shared state in an adaptive manner such that the distribution reflects access patterns of simulating nodes. The algorithms are evaluated within the implementation of PDES-MAS framework using various agent based simulation traces.

ACKNOWLEDGEMENTS

I firstly need to thank my supervisor Dr.Georgios Theodoropoulos who has been very supportive and patient with my research and always kind and helpful whenever I needed. Without him I would not have the encouragement and enthusiasm to even think of pursuing research for more than 3 years. I also want thank my thesis group members : Dr.Iain Styles and Dr.Peter Tino for their invaluable advise and support towards my research.

I am personally indebted to Dr.Robert Charles Minson and Dr.Bart Craenen for their ideas, insights and inspirations towards my research. For the last five years i have enjoyed the time with my friends and little distractions when playing football or tennis or even talking politics: Ariff, Rodrigo, Osama, Sarah, Hasaan, ... I also would like to thank Dr.Dan chen, Dr.Ton oguara and Dr.Michael Lees for their help and advise during my master's dissertation.

Finally to the people I would miss the whole world for : Mom, Dad, brothers' family and my wife for their unfathomable support and love throughout my life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Simulations are commonly used to replicate or emulate real-time scenarios for understanding and studying different building models such as air traffic control, advanced telecommunication network. Computer simulation is part of a program which simulates a real or a physical system. Computer simulations are increasing becoming popular for their feasibility, efficiency and cost effectiveness. They are used to simulate larger and complex scenarios at a lower cost of testing real world systems. Simulations provide a benchmark or a prototype to embark on developing a real physical system. However it is not always possible to test all potential scenarios but results from such experiments would provide ideas and precautions to be taken. There are several applications of computer simulations in various fields such as entertainment, science, technology, business planning and so on. Historically, two types of simulations exist namely *analytic simulations* and *virtual environments* [37]. Analytic simulations aim to run simulations 'as fast as possible' exploiting computer resources to distribute a simulation system[37]. Systems developed in early 90' s based on analytical simulations such as SPEEDES [133], Warped [87], HLA[54, 55] (though used in both analytic and virtual environments) focussed on performance, scalability, throughput and inter-operability of the simulation systems. On the other hand, virtual environments provide a human interactive *virtual world* (a computer generated simulation world) to participate and interact within the simulation system. Earlier implementations of such systems used in military applications for combat training (SIMNET

[2]) and so on. Another common example is flight simulator for pilot training. It uses sophisticated hardware and software to train pilots under different scenarios of weather conditions and flight conditions. Recently, such simulations are used commercially in gaming such as World of Warcraft [15] and even in sports such as Formula 1 racing driver training. However, simulation systems are rather becoming more complex that require an environment for users in thousands, or even in millions to inhibit and interact with each other in the simulation world.

In this thesis, the focus is towards the influence of computer simulations on developing complex systems such as *Multi-Agent Systems (MAS)*. Simulation has a key role to play in the development of agent-based systems and help researchers to understand the implications of alternate architectures[79]. Different notions and definitions of an agent exist in the research community but in general 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 via message passing [150]. Thus, agent based system is a complex, autonomous system which consists of an environment with several agents and objects that can co-exist and communicate with each other. More often the environment of MAS is assumed to be dynamic and non-deterministic. Common applications of agent based systems include control of mobile robots, computer games, telecommunications and military applications. As simulation systems become larger and complex, scalability becomes an issue for their successful deployment. The requirements imposed by these simulations far exceed the computational capabilities of general Von Neumann architecture. Over-provisioning hardware to address the scalability might not be a solution because it is not viable and affordable. A generation of companies used so-called *Super-Computers*, that are intended for speed of calculation and high throughput solved some of the issues. Computer simulations with complex mathematical models fit such system such as weather forecast, climate research and so on. Some existing systems are Fujitsu K Computer[38] and IBM RoadRunner [52].

An emerging viable approach is parallel execution of complex MAS model on low cost

and high performance computing resources commonly known as *distributed simulations*, all tend to solve or model the problem at hand. The idea of such simulation is to ensure four properties such as reduced execution time, distribution of simulation state, integrating simulators running on different hardware and fault tolerance [37]. There are several approaches proposed over the years to execute MAS models on a distributed environment [140]. All approaches have their limitations and applicability of MAS models. One such approach decomposes the simulation model into non-overlapping regions and execute over available computing resources [140]. A simple example would be modelling an air traffic control system where each process simulating an airport executing several actions such as arrival and departure of passenger and cargo flights. The interactions between simulating nodes such as sending and receiving information of flights, passengers, cargo and so on are modelled as messages.

Another approach for scalable solution is to distribute the shared state (environment) of the simulation model among a cluster of nodes either physically located within a building or located geographically at different locations. PDES-MAS (Parallel Discrete Event Simulations for Multi-Agent Systems) is one such approach specifically designed to support large scale MAS models [140]. More detailed description of the PDES-MAS framework will be presented in chapter 3. In summary, PDES-MAS framework uses techniques from PDES (Parallel Discrete Event Simulation, for more details see section 2.6 in chapter 2) to distribute and run parallel simulation of multi-agent systems. Based on PDES paradigm, 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 PDES-MAS simulation, *Agent Logical Processes (ALPs)* model agents behaviour and the environment of a simulation system is maintained by a network of tree-like structured server LPs called *Communication Logical Processes (CLPs)*. CLPs implement a distributed shared memory (DSM) structure whereby publicly accessible variables are represented by *Shared-State Variable (SSV)* data structures which maintains the history of values taken by a particular variable over time [140]. ALPs also provide an in-

terface for the user simulation to communicate with the framework. In response to the sensing and acting of agents in the simulation, ALPs perform reads (queries) and writes on SSVs in a (logical) time synchronised manner [24].

1.1 The problem

A very large scale data structures are required and used more than ever before in the deployment of large scale distributed simulation systems in different applications such as sensor networks, interactive media, Voice-over-IP services, Content Distribution Networks, Peer-to-Peer systems, Distributed Virtual Environments, massively multi-player online games and distributed simulations. Applications such as Skype [129], Yahoo Messenger [151] are examples of such large scale systems. 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 data access. 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. One of the problems in accessing data is to define an *Interest Expression (IE)* for a system to access interested or required data. An interest expression is a method of expressing an area of interest to the simulating entity. A survey of different forms of IEs and their applications is presented in chapter 2.

Within this context, our focus is towards query based access mechanism which, in general, has two types namely *ID Query* and *Range Query*. An ID query accesses the value of a variable whereas a range query accesses a set of variables that match a range predicate. When a system handles large sets of data with users accessing the shared data asynchronously, the mechanisms required to manage shared data across distributed resources accurately and up-to-date is non-trivial [140]. It should be scalable to sustain

hundreds or thousands of user accesses.

Several approaches dealing with data distribution management and data access in Distributed Memory models, Distributed Virtual Environments (DVEs), distributed simulations and so on are presented in chapter 2. This thesis implements a notion of synchronised Range Query within the PDES-MAS framework. A Range Query (which is a form of *Interest Expression (IE)*, see section 2.1 in chapter 2) expresses an interest on a set of variables with values that match a range predicate. A simple form of range query can be defined as *range_query(min_val, max_val)*. A range query can be used on N-dimensions where the range can be expressed for each dimension. The expression is analogous to defining a sensor range of an agent within which all objects inside the range can be accessed. But the problem addressed in this thesis is more complex because PDES-MAS being implemented based on PDES paradigm has to ensure that the events (accessing shared state) are processed in a time ordered fashion. There are plethora of approaches reported on the problem of synchronisation (ordering the events) in PDES system (see chapter 2). But the complexity is due to the implementation of SSV data structure within PDES-MAS framework. A SSV data structure maintains different values at different (logical) time periods of the simulation. Being an optimistic PDES (see section 2.6 in chapter 2) kernel, a history of values of SSVs evolved over time is maintained. So, mechanisms required to validate range queries (issued by agents) in a time ordered fashion over distributed data are non-trivial. The problem is analogous to *Interest Management* approaches as this thesis also provides provisions to deal with the problem of distributing data in an adaptive manner within the implementation of PDES-MAS framework. The aim is to localise data access to reduce service latency and execution times of the simulation.

1.2 Contribution

This thesis presents a notion of synchronised range query within the PDES-MAS framework. PDES-MAS has been initially developed with the assumption of universal knowl-

edge of IDs of state variables in the simulation system. It provided ID query based access mechanism to access data using SSV data structure. But this thesis provides mechanisms to

1. locate a set of variables that match a range predicate in a time synchronised manner,
2. reflect the changes in values of SSV data structure in a time synchronised manner and
3. migrate state variables in an adaptive manner based on the access patterns of the simulated agents.

So, my contributions include developing algorithms within PDES-MAS framework to improve the scalability of the system. The algorithms are

1. Synchronised Range Queries algorithm – The algorithm provides mechanisms (concept similar to SSV data structure) to access a set of variables that match a range predicate in a time synchronised manner. This requires maintaining the range information of all SSVs (in time periods) that can be accessed over the CLP tree. The algorithm also provides mechanisms to rollback and delete the existence of a range query from the CLP tree in a time synchronised manner.
2. Synchronised Range Updates algorithm – The purpose of the algorithm is to update the topology with the changes in the values of SSVs in a time ordered fashion. Instead of flooding to all CLPs, it provides mechanisms to propagate an update (only if required) through neighbouring connections of CLPs. Such update propagation could invalidate and rollback previously issued range queries (in the sense range queries arrived earlier in real time).
3. State Migration algorithm – The aim of this algorithm is to achieve locality of data access in an adaptive manner. The algorithm provides mechanisms to monitor and migrate a subset of SSVs towards the agents that access the most.

My other contributions towards the development of PDES-MAS include

1. Integration of PDES-MAS kernel – Though the design and implementation of PDES-MAS framework has been established before, the framework has never been integrated to function as a parallel simulation kernel. Towards that direction, my contribution is on testing and integrating the functionalities and working of the framework.

2. MWGrid: Use-Case – MWGrid (Medieval Warfare on the Grid, see chapter 7) project seeks to address the problems of military logistics of the Battle of Manzikert in 1071 AD using agent based modelling and distributed simulations using PDES-MAS framework [100]. My contribution is towards the development of an *Interface Layer* that translates the communication between MAS models and PDES-MAS framework. The work is still in progress to run very large scale experimentation but an initial analysis has shown promises towards that direction.

The structure of the thesis is as follows. A survey of different Interest Expressions (IEs) and various interest management approaches to optimise data access in different fields and applications is presented in chapter 2 including an overview of PDES paradigm, MAS and modelling MAS into discrete models; chapter 3 presents an overview of various concepts behind the design and implementation of PDES-MAS framework; algorithms and data structures implementing time synchronised range queries and updates within the implementation of PDES-MAS is presented in chapter 4; state migration algorithm to adaptively localise data access is presented in chapter 5; an evaluation of the system with a benchmark and scalability analysis of PDES-MAS kernel is presented in chapter 6; an overview of the design and architecture of MW-Grid (a Use-Case simulation toolkit) is presented in chapter 7 and finally the conclusion and possible future work is presented in chapter 8.

CHAPTER 2

BACKGROUND AND RELATED WORK

This thesis presents a notion of synchronised range queries within the implementation of PDES-MAS framework. The thesis also presents mechanisms to localise data access in an adaptive manner within the context of distributed simulations. In an overview, the problem comes under the category of data access management, data partitioning and data distribution management but more closely associated with the approaches of Interest Management (IM). So, this chapter presents a survey of approaches related to the problem presented in this thesis in two stages; the first stage covers a survey of several Interest Expressions (IEs) (data access mechanisms) and IM approaches in different applications such as Distributed Memory models, DVEs and distributed simulations with an overview of each domain and its applicability. The second stage presents a survey of synchronisation and load management mechanisms within the context of distributed simulations.

2.1 Interest Expressions

This section presents a survey of interest expressions used in several applications of distributed systems and its relation to the range query expression within the context of distributed systems. An *Interest Expression (IE)* of a simulation entity is an expression of interest towards neighbouring entities in order to interact with them correctly[99]. For example, a simulation entity like a tank in a war field would require knowing the presence

of all objects in its surroundings such as rocks, tanks, civilians and soldiers within its radius (visible range). The expression could even be filtered to receive only specific objects that could cause imminent danger to the tank. IEs take different forms and in general, there are four common expressions in the literature to specify an entity' s area of interest. They are,

1. Formulas – The area of interest could be expressed as a complex form of mathematical *formulas* (a simple example would be using distance or radius formulas),

2. Cells – A simple form is to divide a space into cells (regular or irregular sized) where objects subscribe to interested cells. The trade-off, here, is the size of a cell which depends on the application.

3. Range Extents – An extent is typically a range window on a N-dimensional space. The idea is that a user subscribes to a range predicate and overlapping range window updates to the user.

4. Auras – An *aura* is a subspace in which interaction may occur. It is analogous to a sensor range. The idea is that updates are sent when auras of interested nodes overlap with each other.

The better form of IE is a mathematical formula as the expression can accurately define a region or continuous or irregular shapes. The remaining expressions such as cells, range extents and auras are used in different forms to extract data. All forms of IEs are more or less analogous to using the notion of range queries. The range query expression used in this thesis is similar to the expression form *range extent* that defines a range window with minimum and maximum range values.

The problem of interest expression and interest management is explained with an example depicted in figure 2.1, in which a radius of interest is approximated using a number of grid cells. The idea is to receive updates from all cells that overlap user's range of interest.

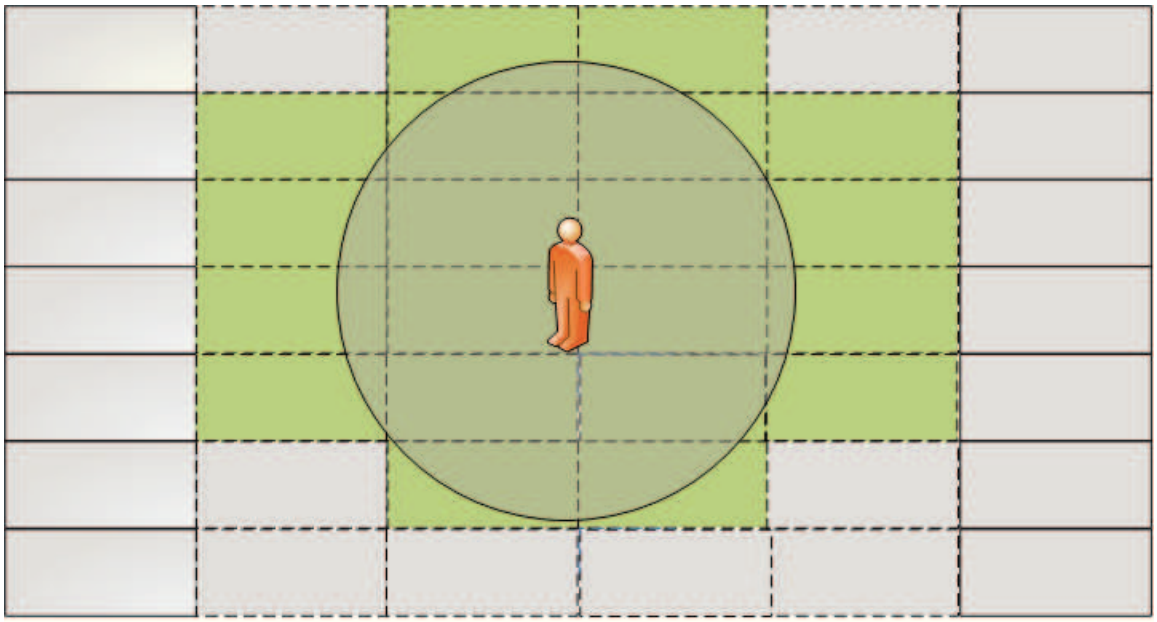


Figure 2.1: The space divided into discrete cells. The range of interest of the depicted person is presented in a circle and its intersecting cells are highlighted in light green colour (figure adapted from [130]).

As the figure clearly suggests that even a partially overlapped region of a cell has to send updates of the entire cell. If the updates are sent quite frequently, it will be an overhead to the bandwidth of the system. More often, IE is approximate but Interest Management (IM) techniques optimises or alleviates some of IE deficiencies. The techniques are mostly developed based on the assumptions and interaction patterns of the entities in an application.

An IE is merely an expression which is translated to an action such as *read* to objects in a simulation model. They are called data access mechanisms. So, a data access mechanism is a read or write to a data object distributed across the system. Nodes send queries, typically messages, to access the required data. The implementation and semantic representation of such simple reads and writes are quite different for each application. It could be based on temporal semantics such as [77], consistency semantics such as [62] or query based semantics [93], but our focus is more towards query based access mechanisms.

2.1.1 Query Mechanisms

Our implementation of data access is based on query semantics, which in general comes in two different forms[1], namely access via *ID-Queries* and access via *Range-Queries*. 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.

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-Query, 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 [107]), to operating systems (e.g. Linda [41]) and application level (e.g. relational databases research [106]. A simple form of range query can be defined as *range_query(min_val, max_val).*

Both range query and ID query can be used interchangeably emulating each other operation[93]. A ID query system can implement a Range-query by performing a ID read for every variable in the system and matching their value against the range window.

[1] contents and definitions in this section are taken from [137]

$$range\_query(min\_val, max\_val) = \{v \in V read(v)(if min\_val \leq read(v) \leq max\_val)\}. \tag{2.1}$$

where V is the set of all variables in the system. On the other hand, a Range-query based system can perform a ID query by issuing a read with the largest range window and match their IDs with the requested data item. As mentioned earlier, the expression of interest may be approximate but IM mechanisms can be used to improve or optimise the performance of the system. The Range-Query expression used in the thesis is similar to the interest expression called range extents, where a range is defined with minimum and maximum values.

2.1.2 Summary

This section presented an overview of different forms of IE expressions such as formulas, cells, extents and auras. The implementation of an IE depends on the design and methodology of applications. Within distributed systems, these expressions are translated into simple reads and writes messages. However, the usage of such simple instructions can vary in semantics and implementation. Broadly, the semantics of access can be based on time (real or logical) or sequence (instructions ordering) or query based. Our focus is more towards query based access mechanism which are of two forms, ID query and Range Query. Most of the forms of IEs are analogous to ID query and range query access mechanisms. In the following sections, a survey of approaches using different forms of IEs and IM approaches is presented.

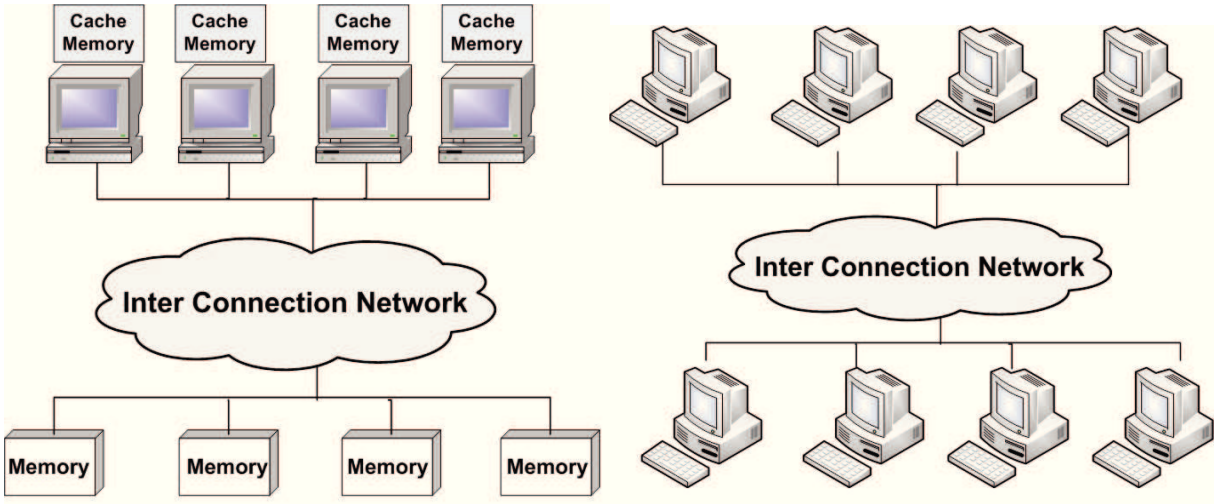
2.2 Stage 1 : Shared and Distributed Memory Systems

Shared Memory Multiprocessors (SMM) is a set of processors accessing concurrently a same set of memory address space. The operation on a shared memory is mostly analogous to a ID read or Write on a shared variable. Though the system can provide access to a set or contiguous space of memory, it does not support range queries[135]. This is because range predicate works on the values or contents in address space instead of actual address space itself.

Two types of shared memory machines are implemented namely Symmetric Multiprocessor (SMP) and Distributed Shared Memory (DSM) as depicted in figure 2.2. SMPs usually interconnect processors to access a shared memory module through a switching device such as bus. *Cache memory* is attached with each processor storing frequently accessed pages to avoid bottleneck of processors accessing the *main* memory. Consistency protocols[135] are required to keep cached pages up-to-date with main memory. The frequency of updates is configured by the programmer weighing trade-off between performance in access times and consistency. Uniform Memory Access (UMA) guarantees average access times to any memory location is uniform. This type of implementation is useful for real-time applications such as parallel processes[135].

Another type of SMM is non-uniform memory access (NUMA) machines which address the issue of access times by moving certain memory closer to processors[135]. It enables faster accesses to *local* memory and slower accesses (when required) to *remote* memory. This is generally used on the requirement to partition and distribute physical memory. The analogy is similar to Distributed Shared Memory machines. Caching in NUMA is an overhead and several cache coherency protocols are proposed at hardware or software level to address the issue.

A Cache-Coherent NUMA [135] system has the task of managing accesses between local cached memory and remote main memory. There are two cache coherent policies to maintain cache consistency as presented in [135]:



(a) Shared Memory Architecture (b) Distributed Shared Memory Architecture

Figure 2.2: Memory Architectures adapted from [37].

- Write Invalidate – A processor writing on a cached memory invalidates all other copies of that page. Reads to a valid copy is satisfied with a local cache memory. If its invalid, it must fetch the valid copy from the main memory.
- Write Update – Write on a cached memory updates all other copies of that page. Reads on a cached page is satisfied locally.

2.2.1 Interest Management in Shared and Distributed Memory Systems

As presented in the previous section, two classes of shared memory systems are implemented such as Symmetric Multiprocessors and Distributed Shared Memory. Cache coherency protocols address the issue of maintaining up-to-date cache memory for processors but fails with access times and performance of the system in real time applications. Several approaches use access patterns of executing programs to address coherency problems. Such approaches generally falls into the category of IM problems addressed on both classes of shared memory systems at the hardware level.

Within SMP, the communication between processors achieved through switch-based connection such as bus which broadcasts messages to all processors. *Snoopy Caching*

protocol[59, 30] is proposed based on write-update mode (see previous section 2.2) caching model. The idea is that writes are updated to all processors with each cache page maintaining a counter. The counter is decremented for each update and reset to n if there is a local read from a processor. The essence of this idea is that if there is no read access from any processor to cache, updates are ceased.

This approach will not be suitable for DSM systems where processors interact through unicast/multicast communications. For that purpose, *directory schemes*[14] are proposed. It employs two models namely, *replication* and *migration*. *Replication* copies updates to all processors cache and *migration* moves pages to processors that access the most. Both uses different strategies to cease and reinstate updates to cache memory. In case of *replication*, counters are used to determine replication based on access patterns on local and remote memories. On the other hand, *migration* is initiated if the cost of access by a processor is twice the cost of migration itself. Approaches proposed on both systems use access patterns to optimise access times of processors. The problem with these approaches is the risk of over loading hardware or software. It is too sophisticated and complex to apply these mechanisms to all types of systems.

2.3 Stage 1 : Peer-to-Peer Systems

Peer-to-peer (P2P) systems are part of distributed systems architecture that connect users (peer nodes) across WAN (e.g.., internet) in a transient network to share information or workload of an application. Typically a peer joining an overlay automatically assumes a share of data or workload that depends on each application. On a system level, peers communicate and route messages using IP addresses. But within an application level, peers are identified with their IDs (sometimes hashed). Traditionally, in a client-server model, the server stores and manages all the information and clients forwards queries to the server. This naturally poses limitations on number of clients requests and the amount of data stored in a server. On the contrary, peer-to-peer systems address the issue of data

sharing by distributing across peers and indexing them to route queries to appropriate peers. The routing mechanisms are of two types namely, unstructured and structured.

Several unstructured systems such as Napster[32] attempts to route queries to appropriate peers with the maintenance of centralised index. Any peer joining the overlay knows the identity of centralised index node. This index node maintains a directory of all available nodes in the network. Gnutella[1], on other hand, uses a distributed index approach to route queries. It maintains routing information of neighbouring nodes and all queries are forwarded via those nodes. KaZaA[78] uses a hierarchical routing approach to route queries from a peer to a super-peer node. A super-peer node is a nominated group leader which maintains the index of other peers within the group and also its neighbouring super-peer nodes. If the request is not satisfied within its group, it forwards to neighbouring super-peer nodes. All of the above mentioned approaches simply follow flooding mechanism to satisfy their requests. There are several issues within such system such as scalability, query latency and handling failures. The main problem with these approaches is that there is no guarantee that a request will ever reach a destination peer within an affordable time. Structured P2P systems, on the other hand, provide a network topology that guarantees routing queries to a destination peer with a minimum number of hops (nodes). Data is distributed among peers in a controlled manner such that subsequent queries are forwarded efficiently. Here, the focus is on the usage of notion of Range Queries in P2P systems and are commonly classified based on the overlying structure that is presented in the following sections.

One family of P2P system approaches use Distributed Hash Table (DHT) to distribute data uniformly across live peer nodes and consistently guarantee routing queries to a destination peer. DHT stores $(key, value)$ pairs, similar to a hash table, to store and retrieve data from the peer nodes in a network overlay[60]. The idea is that each data object is assigned with a unique identifier (taken from a set of key space identifiers such as 160-bit space strings) called key. Keys are mapped on to the network overlay using an algorithm chosen by the application. A simple scheme is to partition the key space uniformly among

peer nodes. Then a routing mechanism is used to consistently retrieve the value associated with a key. Typically, each node maintains a table listing its neighbour node IDS and IP addresses. The queries are forwarded in a progressive manner to the node that is *closer* to the key in the identifier space[60]. The aim is to retrieve data with *O(log N)* hops where N is the number of live peer nodes. Different approaches of DHT mechanisms have been proposed such as CAN [110], Chord [136], Pastry [144] and Tapestry [152]. A short summary of each of these approaches is presented here.

2.3.1 CAN

Content Addressable Network (CAN)[110] is designed as a self-organising and de-centralised P2P system which employs hashing mechanism to distribute data across live peer nodes. The approach supports d-dimensional coordinate space which is divided dynamically among available peers. Each peer node maintains a routing table of IP addresses and virtual coordinate zones of its neighbours. The hash table containing *(Key, Value)* pairs is maintained at a point (Key) using a uniform hash function. Searching an arbitrary point d from a peer is routed along its neighbours to reach the node that maintains the zone in which the point resides. Hash pairs are replicated among k peers to avoid single point of failure and data availability. It has been used in large scale storage management systems and several schemes adapted this approach in virtual reality and online gaming systems[60]. The routing performance of CAN is $O(d)(N^{1/d})$, where d is number of dimensions and N is the number of peers. Figure 2.3 depicts an example of new peer X joining the overlay choosing a random point splitting the zone occupied by node Z and is notified to all neighbours.

2.3.2 Chord

Chord[136] uses a consistent hashing to assign data to peer nodes. It adopts SHA-1 hashing scheme to hash a node IP address and data key. Each peer node is assigned an

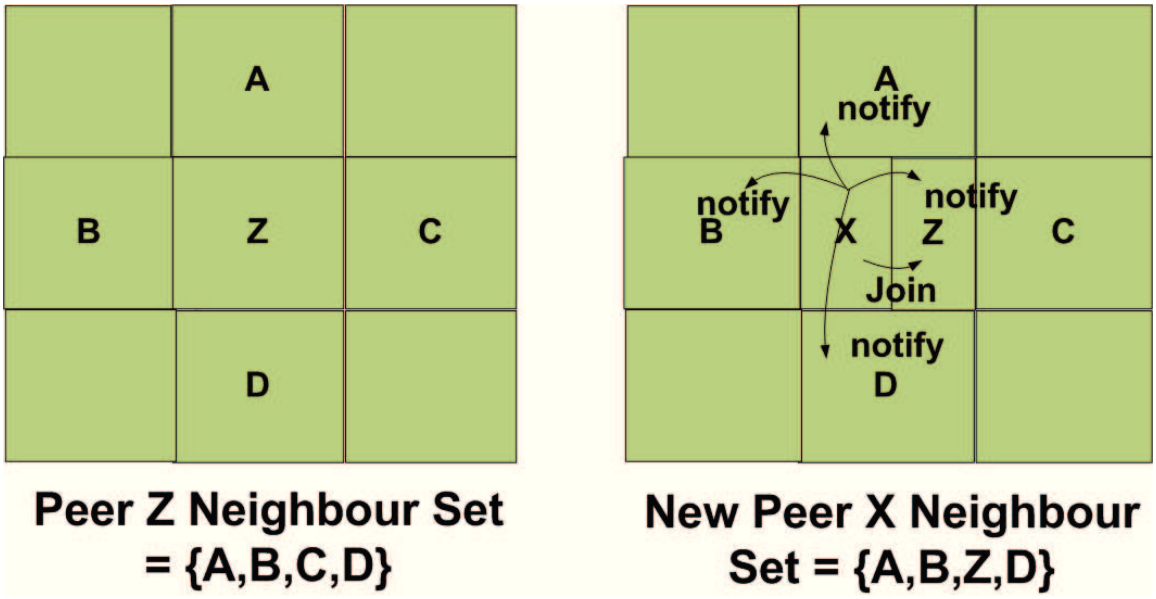


Figure 2.3: CAN overlay (adapted from [110]) before (at the left) and after (at the right) peer X joins the overlay.

identifier m using its IP address and the data key is hashed and stored in a peer node. The load balance is achieved by roughly assigning same number of keys to peer nodes. Each peer node knows its next neighbour node called *successor*. In addition to that, each peer node also maintains a finger pointer table which has m entries, where m is the bit length of node identifier. Lookup of a data from a peer is routed along the circle using routing tables of its successors and response is traversed back along the same route. An example of such routing mechanism is depicted in figure 2.4. Peers joining or leaving the overlay needs to update successor and finger pointers in the overlay to keep tables up-to-date[60]. The approach is quite robust in handling failures and used in several applications such as DNS for efficient and reliable lookup service[60].

2.3.3 Pastry

Pastry[144] uses a plaxton-like prefix routing mechanism to build P2P overlay. Each peer node is assigned a 128-bit identifier and arranged in a ring structure. The node ID is generated randomly and uniformly distributed in a 128-bit hash space. The routing mechanism is based on identifying nodes that is numerically closest to a given key. For

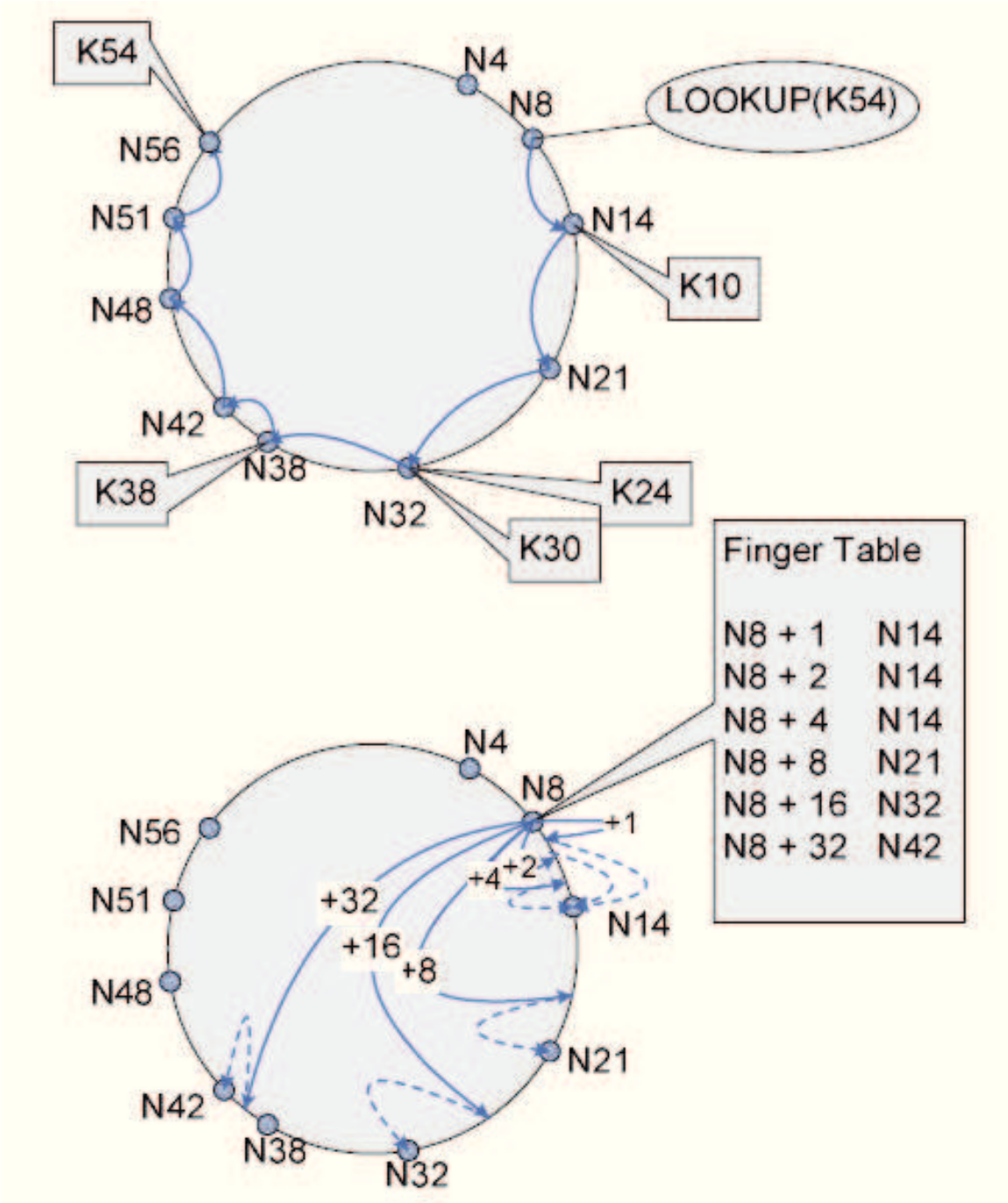


Figure 2.4: Chord ring overlay (taken from [60]) (at the top) depicting a query lookup path through the ring and (at the bottom) depicting the finger table for peer N8.

example, a peer with identifier 1234 searching for a key 4567 would route the message to the node that has first two bits numerically closer (45**). The usage of such locality based scheme is used for several IP-based multicast scheme applications such as Scribe[20]. d

2.3.4 Interest Management in Peer-to-peer Systems

Interest Management approaches over peer-to-peer systems are classified as hash-based and non-hash based. Range Queries are commonly used operation to access data over such network overlays. Though the usage of range queries over DHTs is not efficient (as described in the following section) several approaches attempt to address the issue utilising the randomness of data distribution itself. Here, a survey of approaches using range queries over hash-based and non-hash based will be presented.

2.3.4.1 Range Queries over Hash-based

Range queries are often used in many applications to retrieve a set of data that falls within a range window *[min_value, max_value]*. But the usage of such notion poses a challenge for randomised and de-centralised usage of hash functions[60]. In essence, to compute and retrieve a range of values using range queries over DHT based P2P systems, for each value $v \in$ [*min_value, max_value*] (range window), *lookup(hash(v))* has to be computed. If the keys are in a distant location (from the range querying user node) in the overlay, it will increase query latency and become less scalable. An alternative would be hashing range of values but would require optimal partitioning in prior [122]. If the partitioning is too large, it will be an overhead and if it is too small, it would increase number of hops.

Despite these issues, several approaches have been proposed. In [45], the aim is to use local sensitive hashing scheme to assign similar ranges to a node with a higher probability. For example, a peer can submit a SQL query like

```
Select * from Students where course = Phd.
```

For the first time, the query will be forwarded to a source peer node with *key* as 'course'. Now, DHT is used to store the resulting partition of 'course' to a peer in the system. The subsequent queries will then be satisfied with this local node rather than forwarding it to the source peer node. The approach works fine with a centralised server to extract a partition but complex queries requires several partitions which are distributed

across peer nodes. To address this issue, it proposes a hashing function called *Min-wise Independent Permutations*. The aim is to produce nearby hash values (32-bit identifier space) for similar range of values or data partitions. Peers in the system are also mapped to the identifier space using SHA-1 hash functions. It uses Chord[136] like structure to map data partition hash values to peer nodes hash values for lookup and routing. There are two issues with this approach

- First is the assumption that subsequent range queries will have similar range window or values and
- Second is the number of peers required to store different data partitions.

Another approach based on Prefix Hash Tree[117] is proposed, which recursively splits 1D data space into a tree based structure. Each node has a *threshold* to hold a number of items and exceeding it would split the existing node into two leaf nodes. The leaves, thus, form a continuous range of values with each node having similar number of data items. For scalability, these leaf nodes of PHT are mapped onto live peer nodes using consistent hashing of bit strings (otherwise called prefix representing the path traversal in the tree). The randomised hash function ensures balancing the load across peer overlays. The data that are located closer to each other in PHT structure would not be necessarily closer to each other in peer overlay. This means a range query may have to traverse to a distant node which delays query response times.

Though the approaches presented in this section addressed the issue of complex range queries over DHTs based P2P systems, it is always viewed as a work around to hide the fact that DHTs work for specific data lookup. To gain the advantage of randomness and load balancing data across live peer nodes, it compromises the adaptability of issuing complex queries over such system. There is another family of approaches which avoided hash based schemes to get precise information from P2P systems.

2.3.4.2 Range Queries over Non-Hash-based

Research towards hash-free approaches emerged to address the issue of using complex queries such as range queries over P2P overlays. In [29], a non-trivial range queries is supported over structured P2P overlay which provides *O(log n)* search complexity on top of trie abstraction. The idea is that the usage of tries keeps semantically closer data items are clustered together. This is in sharp contrast to standard DHTs overlays. Implementations of data structures such as Skip List[113] is used to randomly distribute data and connect through a balanced tree with sparse linked lists. At level 0 (lower level) of a skip list contains all nodes in the increasing order connected via doubly-linked list. At each level > 0, each node participates in level i with some probability p. The idea is that higher levels act as *motor ways* to traverse the nodes quickly[113]. The target is to construct the list with search traversal cost (or hops) in *O(log n)*. An example of such a list formation is shown in figure 2.5.

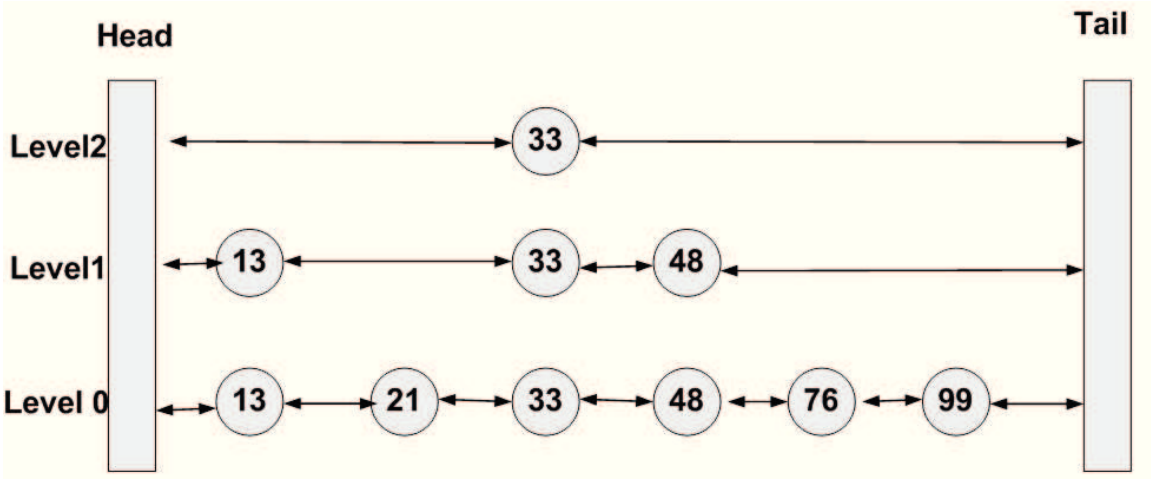


Figure 2.5: A Skip List (adapted from [113]).

To find an item, say element 76, start with higher level and traverse through the lower level lists until it finds the exact node with the element. The problem with this approach is single point of failure. Extension of this approach to distribute and avoid single point of failure is implemented as Skip Graphs[5]. Instead of maintaining a single linked list at each level, each node at level i will be a part of many linked lists to handle failures. Also

each node maintains *O(log n)* neighbours on average. Insert, search and delete operations are similar to Skip Lists but the difference is that there is a lesser chance of overloading a node with queries. The nature of this approach is to maintain nodes in the increasing order making it viable for executing complex queries such as range queries. But unfortunately, the efficient execution of range queries is compromised with load imbalance (due to lack of complete randomness like DHTs).

Apart from approaches using DHTs and Skip Lists, Mercury[12] proposed a design to support multi-attribute range queries and explicit load balancing. The nodes in the system are grouped into hubs that maintain query attributes. Nodes within a hub are arranged in a ring structure maintaining a range of values. The routing mechanism within a hub uses k-long distance pointers to remote nodes. A range query is forwarded to a node which matches the range window. This approach also supports queries with non-uniform range values. An example of 2-D query issued at Mercury overlay is depicted in figure 2.6. The advantage is that queries with multi-attributes can contact any one attribute hub to access the rest of the nodes.

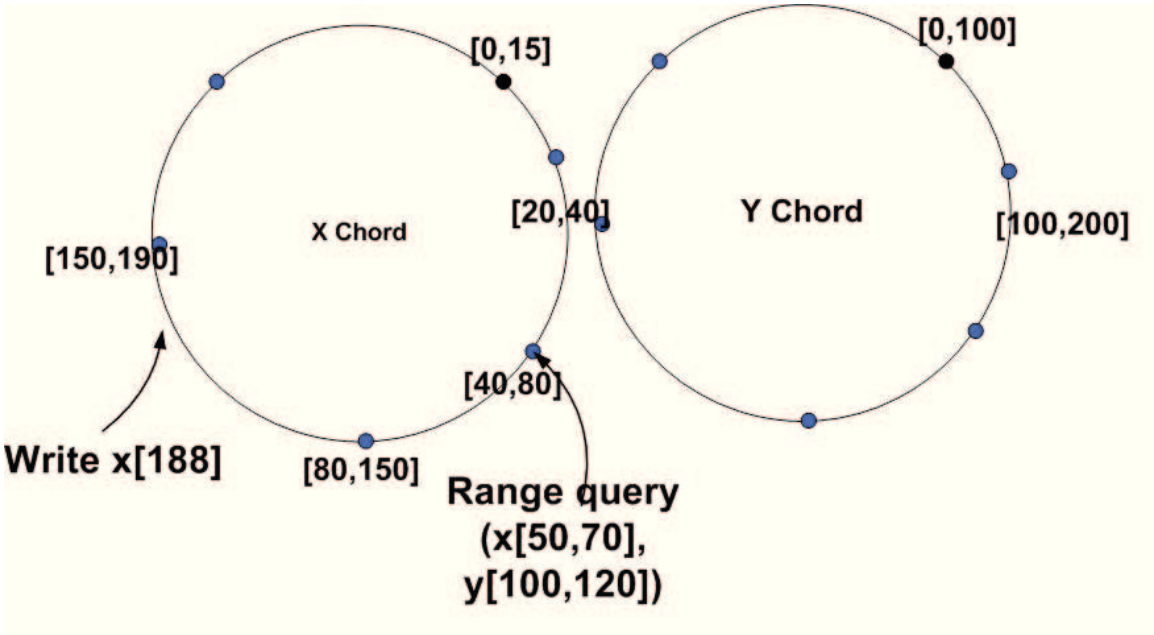


Figure 2.6: Routing queries and updates in Mercury overlay (adapted from [12]).

2.4 Stage 1 : Distributed Virtual Environments

A virtual environment is a part of computer generated simulation world into which human interaction can be embedded. A common practice in aviation industry is flight simulation for pilot training. The common goal is to achieve a more *realistic* environment to the participants (for example, to simulate different wind speeds or climate conditions for a pilot). *Distributed Virtual Environment (DVE)* enables users or participants connected through a network (LAN/WAN) to interact with the simulation world. Research in DVEs has grown significantly mostly because of the requirements of online games. The number of participants is increasing in thousands (sometimes in millions) which demands high bandwidth and low latency network connection. In comparison with analytic simulations, DVEs are mostly real time applications where the requirements of synchronising algorithms are relaxed. For example, if two or more events occur so close for participan's perception, the simulation can deicide the ordering of events without compromising its goals.

Earlier systems of DVEs focussed on military simulations such as combat missions, battle engagements. SIMNET[2] is developed as a human interactive simulation world, in a real time network and interconnected with autonomous simulators. It is developed by Defence Advanced Research Projects Agency (DARPA) and used for battle engagement simulation and war-gaming. It is based on server-less distributed architecture in which users can join and leave at any point of time in the simulation. The disadvantage is that updates are broadcasted to the network and receiver is responsible to filter such updates. Applications developed based on SIMNET are ALSP (Aggregate Level Simulation Protocol)[148] that focussed on aggregating US military and navy simulations for simulating a joint combat operations. DIS (Distributed Interactive Simulation)[47, 104], an extension of SIMNET protocols, addresses the issue of interoperability between different types of users and simulators (for example, tanks, missiles and so on). Several large scale systems are developed based on DIS protocols to support scalability such as NPSNET[83]. However, current military simulations is based on HLA [55, 54] technology

which aims to provide general architecture and reuse of simulations. A *federation* refers to a distributed simulation consisting of several federates where each simulator is a *federate*. HLA supports any type of simulations such as real-time[147], non-real-time(analytical simulation)[70] applications, commercial and so on.

Systems presented so far focussed on scalability and interoperability of large scale simulation systems. Several other research conducted on systems involving small sized participants but focussed on clarity and rendering of visual objects for human perception such as RING[39], SPLINE[25], DIVE[19] and MASSIVE-1[44]. Their goal is to maintain consistency in data across participants using replications or dead reckoning mechanisms (prediction technique based on object information such as position, speed and so on) and reduce bandwidth using several techniques (more on this explored in section 2.4.1). Recently some interest have been shown towards online gaming and multi-player computer games as reported in XPILOT[134], DEE[112], MIMAZE[76] and [23].

In summary, research towards DVEs predominantly focus on human interaction and visual representation. In general, the underlying architecture is mostly based on three types; a simple client/server, clients interacting with distributed servers and server-less (analogous to peer-to-peer systems). The design and implementation largely depends on the type of the application or simulation system. The emergence of online gaming has presented with different challenges in terms of scalability and data consistency.

2.4.1 Interest Management in Distributed Virtual Environments

The tolerance of receiving delayed updates in real time applications such as DVEs is very minimum. Each node (simulating a participant or an object) requires a consistent data to render virtual object with more accuracy. Earlier systems addressed the issue with broadcasting or point to point unicast updates mechanism to the simulation nodes such as SIMNET[2], DIS [47]. One would require sophisticated software or hardware to run such large scale simulations. The emergence of massively multi-player games (MMG) imposed restrictions on bandwidth and hardware (typically computer system

used by a participant). The essence is that disseminating updates only to relevant nodes. The relevance, here, means *interest* expressed for a subset of a information, commonly termed as *area of interest or aura*. It is analogous to the notion of range-queries. It can be associated with sensing capabilities of a user. Within DVE applications, the notion of interest is expressed in different forms which is are broadly classified with their applications.

2.4.1.1 Large Scale Distributed simulations

A large scale distributed simulations in a virtual environment typically includes entities or nodes connected in a network such as LAN/WAN, typically the internet, varying from hundreds to thousands (if not in millions). Each node can be associated with a computer user or a simulation object that can interact with other nodes. Applications of such large scale includes military game training or combat mission simulations. Systems such as SIMET[2] and DIS [47] developed for military logistical training and war gaming that simulate systems minimising the cost of training and testing actual machines or hardware. Data consistency (maintaining same information across nodes or simulating participants) and responsiveness (update latency) are two main issues of real time applications. A survey of several approaches of interest management in such systems are explored in [95]. Broadly they can be classified on their access mechanisms such as *cell-based* (also known as *grid based*) and *region-based* (also known as *zone based*).

Cell-based schemes

Implementation of virtual systems such as DIS[47] consumed large bandwidth (using broadcast mechanism to send state updates) and imposed the requirements of filtering relevant data received by a user. A simple approach is to define a range of interest (ROI) of a simulation entity (like sensor range or visible range or range of radar observing objects) in which state updates are sent only to relevant entities. Cell-based schemes or grid-based schemes (both commonly used for same purpose) split a simulation world into

grid cells. For example, splitting a landscape (terrain) into equal number of cells. The cells are assigned with multicast groups where each cell can be a uniform square shape (as in MOSDAF[124]) or hexagonal (as in NPSNET[83]) or non-uniform grid shape or any random sized grid cell. Updates are disseminated to all entities registered with the multicast address of a cell. In [48], cells are arranged in a uniformly squared shaped grid cells in which any entity wishing to receive an update simply joins the multicast group of a grid cell. In essence, entities can join or disjoin a group based on their knowledge of their location in the grid. An important trade-off is the granularity of a cell which if defined too large which will result in poor approximation of filtering but requires smaller number of multicast groups and works otherwise if defined too small[48].

Though DIS systems broadly use broadcast mechanism to disseminate updates to all entities, several approaches used filtering mechanisms such as multicast groups to filter relevant data. Modular Semi-Automated Forces (ModSAF) [124] is a constructive simulator designed for the Army and fielded in the 1990s which uses filtering mechanisms at the receiver side of an entity. It assigns grid cells which filter packets (messages) based on their types (interested or relevant to the entity). The disadvantage of this mechanism is that individual entities cannot subscribe their interest, instead, the system simulating such entities performs such mechanisms. Naval Postgraduate School NET (NPSNET)[83] (a DIS 2.0.3-compliant 3D visual simulator test-bed developed by the students and researchers at NPS) uses a hexagonal grid cells where an entity subscribes to a set of cells defined by a *Domain of Interest (DOI)* (otherwise a radius of grid cells). Grid cells are assigned multicast groups statically at the start of the simulation. A group leader in the system (oldest active group) has the authority to add or remove entities from a group. Other systems developed based on DIS is reported in [142] and [84] that aim to provide scalable virtual environments and reduce bandwidth incurred on using DIS protocols. Unlike such systems, HLA [55, 54] technology emerged as a generic architecture for both analytic simulations and real time applications. Various approaches developed for Data Distribution Management (DDM) services are presented below.

The goal of HLA DDM [97, 96] is to reduce the message traffic (or bandwidth) incurred on a *federate* (a simulation node). It is intended to provide an efficient, scalable and a generic interfacing DDM services to the processing federates in the simulation. Each *federate* has a subscription region (analogous to area of interest or specifically range-queries) and disseminate updates (analogous to writes) to interested nodes. The idea to support such a system is based on routing spaces. A *routing space* is defined as a multidimensional coordinate system in which each federate defines an interest (either to publish or subscribe) on data (or a region)[130]. A region is a set of extents where an extent is a bounded range covered along each dimension. Each federate defines either an update or a subscribe region which defines a set of routing space coordinate values of an object defining a region[97]. A simple example of such mechanism is depicted in figure 2.7. It depicts a 2D update region (U1) and two subscription regions (S1 and S2) where updates are disseminated to subscribed federates when there is an overlap between subscription and update regions as U1 and S1 overlaps in figure 2.7.

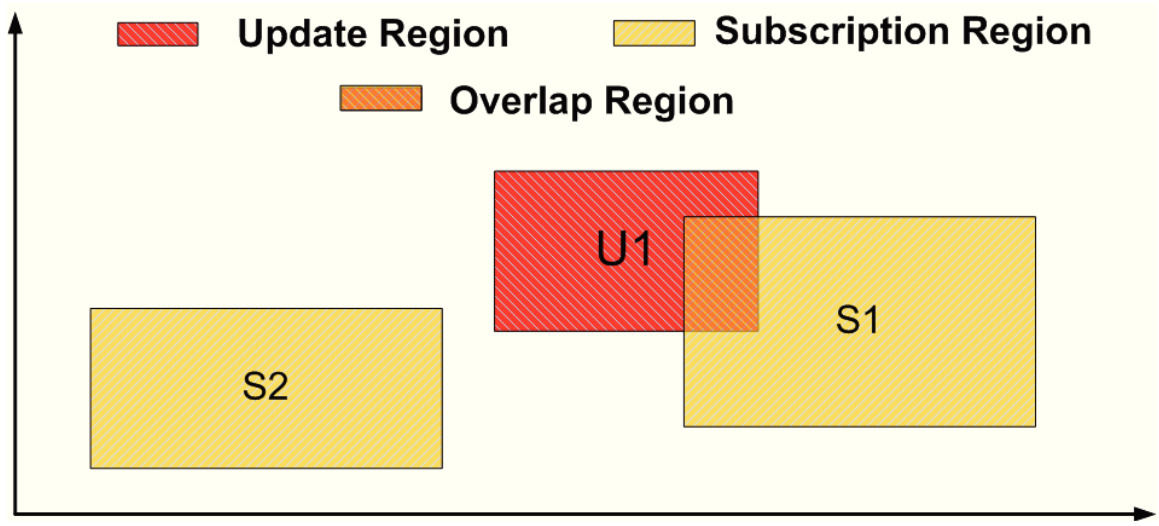


Figure 2.7: A 2D Update and Subscription regions where updates are sent from Update region (U1) to federates that prescribed subscription region (S1) (adapted from [97]).

Such regions are confined to grid cells. Cells are assigned with multicast groups to automatically send updates. An update (or publish) region is restricted to one cell and subscription region can overlap more than one cell. For example, figure 2.8 depicts a two dimensional grid cells (four cells where each cell is numbered at the top left), where cell 1

has a spy plane and other planes are placed in cells 2 and 3. The subscription region for the spy plane covers both cells 1 and 2 and overlap is observed in cell 2. Here, multicast groups are formed for cells 1 to 3 as (MG1-3). Federate 1 (simulating spy plane) will subscribe for multicast groups (MG-1, 2) and other federates (simulating other planes) will publish their groups (MG2, 3).

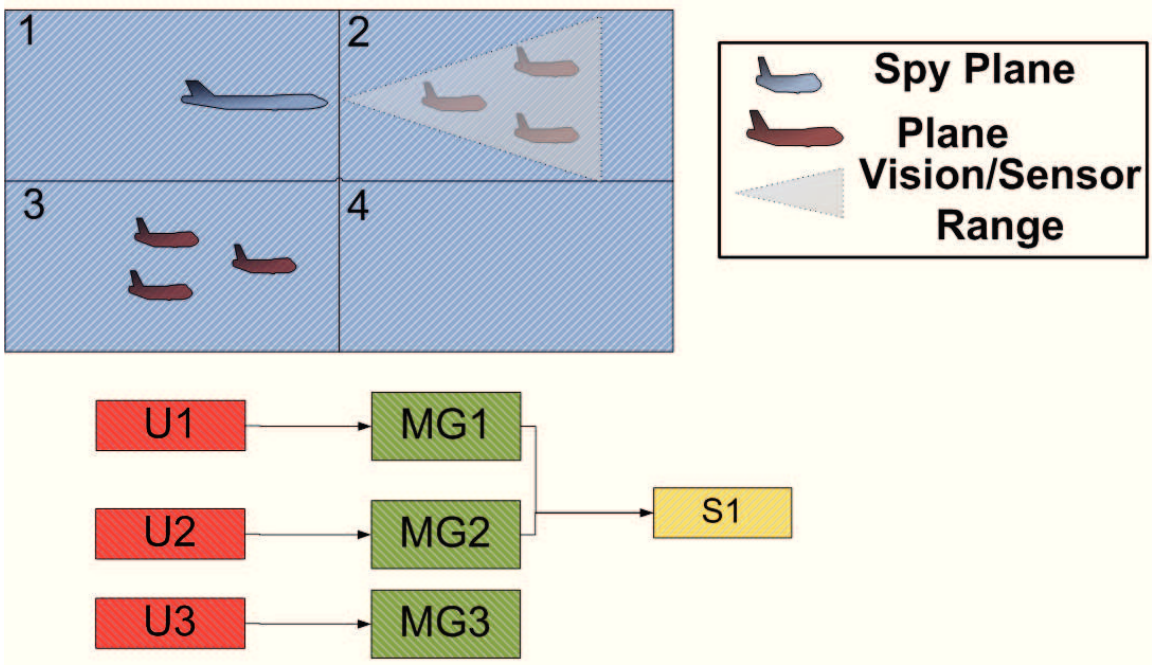


Figure 2.8: Figure (adapted from [97]) depicting 2-D Cells (at the top) with an object's (here, spy plane) interested cells. U1, U2 and U3 are update cells for cell regions 1,2 and 3. Subscription region for the object (spy plane) covers both cells 1 and 2. The corresponding assignment of multicast groups is presented (at the bottom).

The problem with this approach is that in real time applications such as online games, where some objects are expected to be moving all the time, frequency of updates (bandwidth) or delays (network latency) can affect the outcome of the simulation. Monitoring and managing multicast groups in a large scale simulation will be an overhead. Several mechanisms are proposed in [97] to define update and subscription regions larger or smaller than actual size (sensor range analogous to range-queries) to reduce this severity.

Systems developed based on HLA implementation [116, 51] allowed different sized cells (non-uniform grid cells) to assign multicast groups based on the frequency (high/low) of updates incurred in the system. Also, the granularity of a cell is adjusted to assign a

more precise region (according to the object sensor range or range query), if an object is moving slowly. This enables systems to run large scale simulations with mixed sizes of grid cells. Most of the cell based DDM services are based on IP multicast mechanisms which poses the possibility of running out multicast addresses imposed by the requirements of number of groups[99]. This also shows that optimisation techniques are required to assign multicast groups wisely. In [16], a dynamic approach to assign multicast groups is introduced. Based on this approach, a cell is assigned a multicast group if and only if atleast one subscription region and one update region exist on that cell. With the same example presented in figure 2.8, of 3 multicast groups (MG1-3), only MG3 will be created as it creates an intersection of subscription and update regions. This dynamic approach reduces the number of groups drastically compared to the fixed or static approach to assign multicast groups. Similarly, [10] proposes a mechanism to reduce the number of groups dynamically by grouping cells of fine-grained grid into clusters forming a multicast group. Evolving over time, as cells become inactive (no subscription or publishers) they are removed from the cluster. The clusters are reconfigured over time, as cells are added and removed from existing or new groups.

Region-based schemes

Region based approach simply defines a range window (min-val, max-val) to subscribe to an area of interest. A region based DDM approach implemented within HLA[97] compares the subscription and update regions directly in order to find a relevant or matching data. The system supports two types of data filtering such as *class-based and value-based*. In class-based scheme, a federate can subscribe to values of an attribute class that will receive updates of all objects of that class in the entire federation. Whereas in value-based scheme, a federate can subscribe to a specified range of values within an attribute class to filter data based on values. A small federation can utilise the simple class-based scheme to receive updates but as the simulation grows to a larger scale, value-based is more effective in filtering data specific to the subscribing federate.

The problem with region-based scheme is the computational overhead in recalculating changes in publication and subscription regions to all federates. A centralised approach to manage and monitor these global state information would either be a bottleneck or degrading the performance of the system. Morse's thesis [99] presented the problem of assigning multicast groups as a NP-complete problem as the connections between federates are translated as a connected network graphs. The connections, here, represent a set of updates from federates where each update can have a single or multiple messages. Figure 2.9 depicts the problem of optimising multicast groupings. In multicast grouping - 1, updates (U1, U2) from federate 1 are sent to federates 2, 3 and 4. The problem with this scheme is that federates 2 and 3 are receiving an extra update (U2) and federate 4 is receiving an extra update (U1). A similar pattern is observed with the multicast grouping - 2 in figure 2.9.

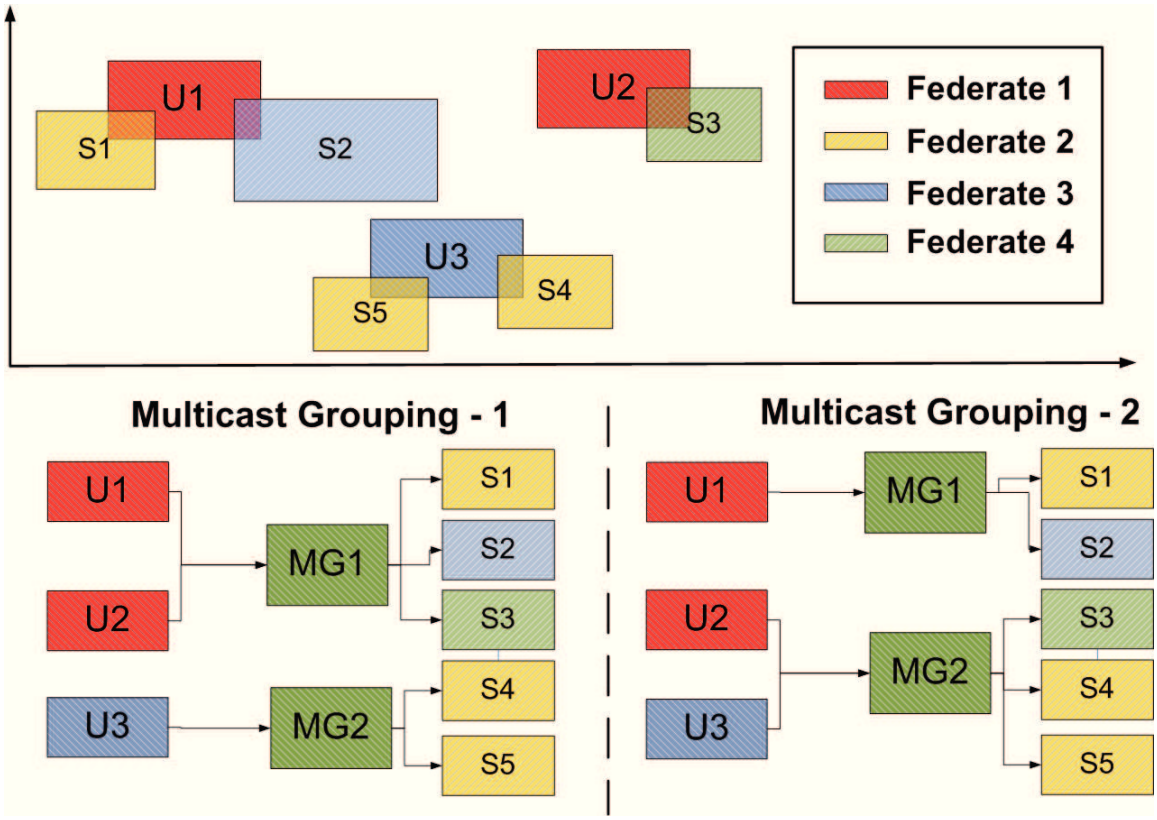


Figure 2.9: Figure (adapted from [93]) depicting possible assignment of multicast groups (shown at the bottom) for update and subscriptions regions (shown at the top).

The problem is complicated by the fact that the publish and subscribe regions can

change constantly over time. The overhead of gathering the information across all federates to determine an optimised multicast grouping can degrade the performance of the system as shown in [98]. In [98], a mechanism called Online Grouping approach uses a minimum subset (local knowledge) of regions and region intersections to form a multicast group based on the available groups in the simulation. It always poses a trade-off between assigning multicast groups (precise calculation of receiving nodes) and cost of collecting global state information (overhead). Even with distributed and local knowledge, the problem always exist on finding an optimised solution in a distributed simulation.

2.4.1.2 Massively Multiplayer Online Games

Online games are used more than ever before commercially and users connect from different geographical locations. The requirement to provide seamless transmission of data with low latency network connection has increased manifold. In [61], the infrastructure to distribute game state is based on P2P overlay *Pastry*[144]. It uses Scribe[20] to build a scalable application level multicast infrastructure on top of Pastry. The idea is that simulated game world is divided into regions and each region is assigned with a node using Pastry's key space. These nodes act as multicast group coordinators and all peers (players and objects) within these groups (logically regions) are updated automatically. The structure is similar to grid cells, where data management exhibits players interests restricted to a region. It also provides a primary backup or replicas for the coordinators incase of failures. *SimMud* system built for this purpose, though showing a good performance, is only experimented in a smaller scale of 4000 nodes[61]. The issue with this system is that it assumes minimal movement of a player in the world to change their groups. So, the granularity of a region should be large enough. Also, the flooding mechanism restricts the number of players participating in the system.

To restrict flooding and enable precise data access required by a user, Colyseus[11] built on a distributed P2P overlay called Mercury[12]. It utilises a simple publish-subscribe system to precisely express a range of interest or otherwise commonly termed range-

queries. Each object is assigned a node which holds a *primary* copy and also maintains a *secondary* copy updated periodically (dependent on application). *Subscriptions* or range-queries are forwarded to some set of nodes using Mercury attribute hubs where they will be stored. Updates are sent periodically published with the latest values in the overlay. This enables to route both queries and publication to meet at some location in the overlay, a rendezvous node, where publications are routed to interested nodes using IP connections. The querying node will also send a *keep-alive* message to keep receiving updates. If connection is terminated, it again uses the rendezvous approach to establish back the connection. A simple example with 1D range query is depicted in figure 2.10.

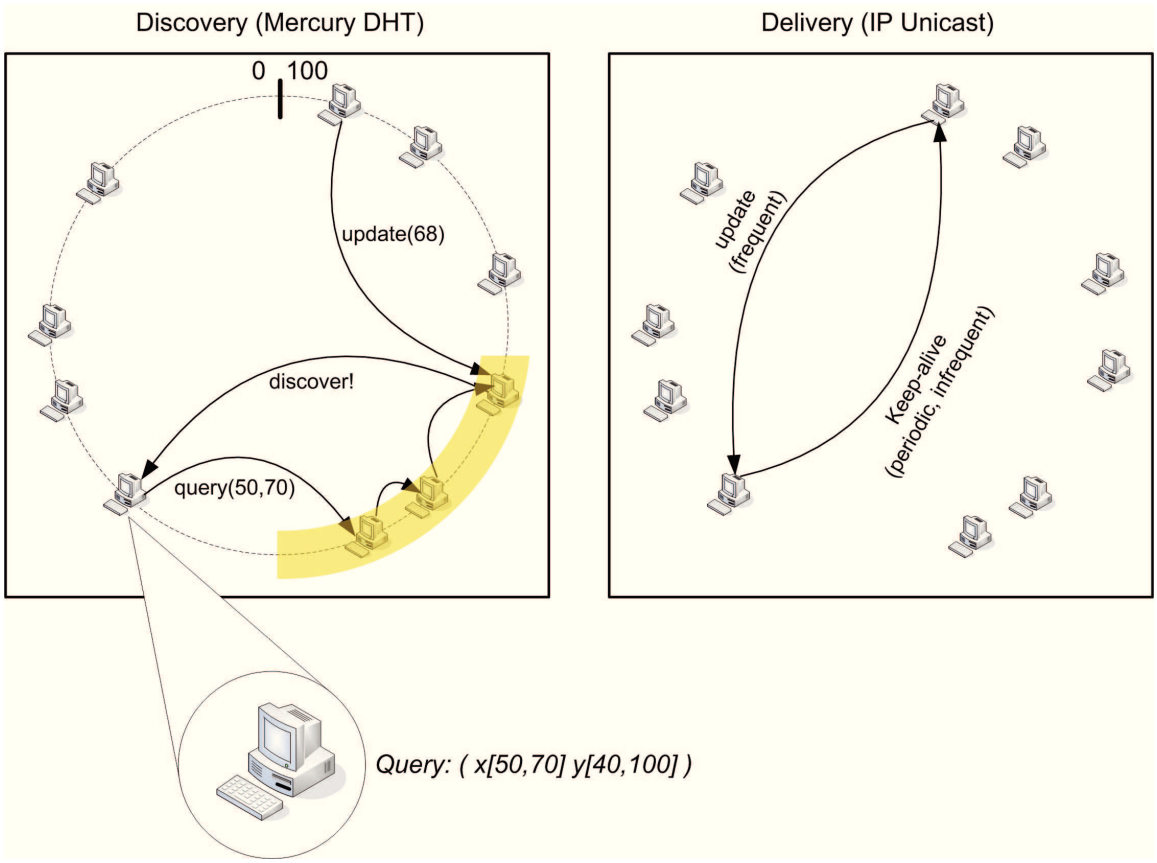


Figure 2.10: Colyseus two level system of discovering and delivering data (taken from [93]).

Approaches presented so far expressed their 'interest' in a bounded range of window and updates are disseminated periodically to the interested peers in the network. The problem still persists to express more precisely an area of interest rather than approx-

imating a range using bounded windows. A family of approaches proposed based on expressing an area of interest called *aura*. It is analogous to the sensing capabilities of a user or simply range queries. In [8], a spatial model of interaction is expressed in terms of *aura, focus* and *nimbus*. An *aura* is a subspace in which interaction may occur. When aura's of users overlap, they can be aware of each other in the environment. Aura is always symmetric as it requires a full bounding rectangle or cells to cover the aura. But an aura is further divided into *a focus and a nimbus*. A focus is a sub space in which the person focusses his attention (see figure 2.11). The more an object is within the person's focus the more aware of it. On the other hand, a nimbus is a sub space in which others are aware of person's activity. The more the object is within the nimbus, the more it is aware of that person. The combinations of foci and nimbi in games for users to perceive night vision (infrared binaculors) and other scenarios (see an example in figure 2.12).

Figure 2.11: A representation of a focus region using a snapshot from Quake game (taken from [53]).

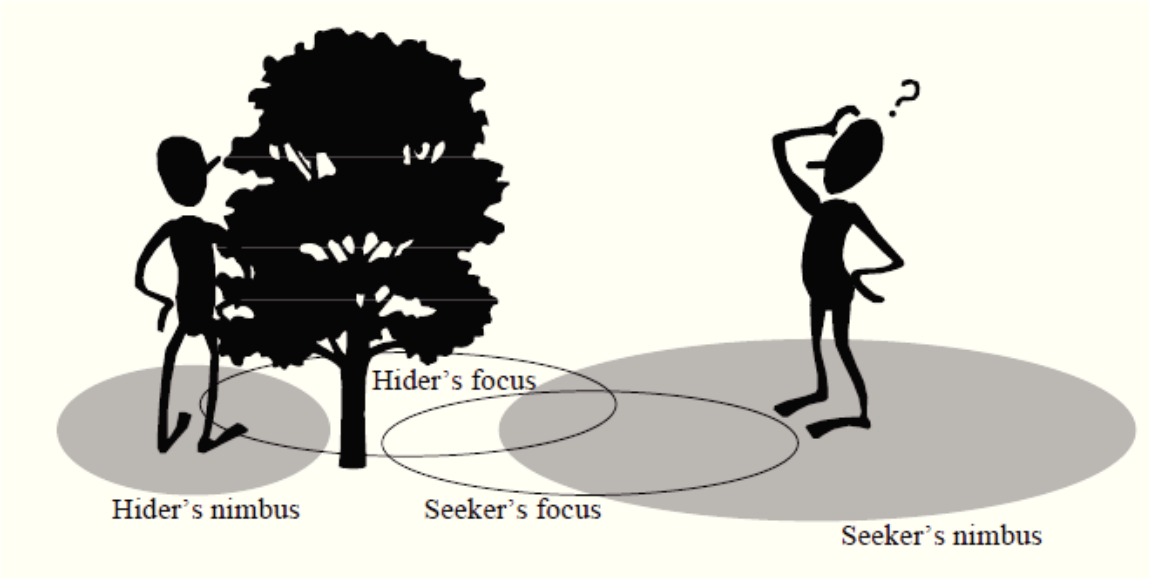


Figure 2.12: Figure (taken from [130]) depicting an example of focus and nimbus using a simple hide and seek game approach.

Earlier attempts to aura based DIVE[19] use the combination of aura, focus and nimbus to replicate shared data across users or peers to maintain consistency. Virtual Society Project [63] is based on a client-server model where it employs aura based interest management scheme to replicate and disseminate updates across a limited number of peers connected to multicast groups. In MASSIVE-1[44], each user is allocated a node and the system uses the aura to identify the scope of its interest. Each group is managed by a manager that detects collisions between auras of users and establish unicast connections between them. Each group maintains a portion of virtual world and they exchange information across groups as and when users require them. The total bandwidth requirement is in the order of O(NM) where N is the number of concurrent users and M is the number of groups. The advantage is that unicast communication is established only when auras overlap and extract precise information. The problem is that the system is not scalable and bandwidth requirements far exceed than normal multicast communication (O(N))[44].

MASSIVE-2[9] addressed the issue of communication overhead taking account of application-related limitations such as boundaries and bandwidth requirements. It used third party object based multicast communication and replication model to update users.

It adapts to the changes to the users auras that either modify or delete existing awareness relationships between users. Both MASSIVE-1 and 2 suffered with the notion of single manager to delicate collision detection and establish communication between peers. MASSIVE-3[114] combined the standard design with the notion of locales[7] taken from the SPLINE[25] distributed virtual reality system. A *locale* defines its own co-ordinate system. This allowed continuous spaces to be modelled as scene graphs of adjacent and interconnecting worlds. In essence, it has different managers to replicate, communicate, render and so on in an appropriate manner.

Morgan [94] proposed an approach using similar ideas of aura-centric routing in which each entity in the DVE defines an aura of influence (AoI). A set of objects within a set are *objects under consideration* if they are considered for collision[94]. A predicted area of influence (PAI) is calculated from the object position vector and potential speed over a given unit time. An entity's PAI simply represents the area that may contain the entity's AoI at some point in the near future. Low frequency PAI messages are sent to predict the future movement of objects. Updates are disseminated when collision detection algorithm detects objects AOI overlap.

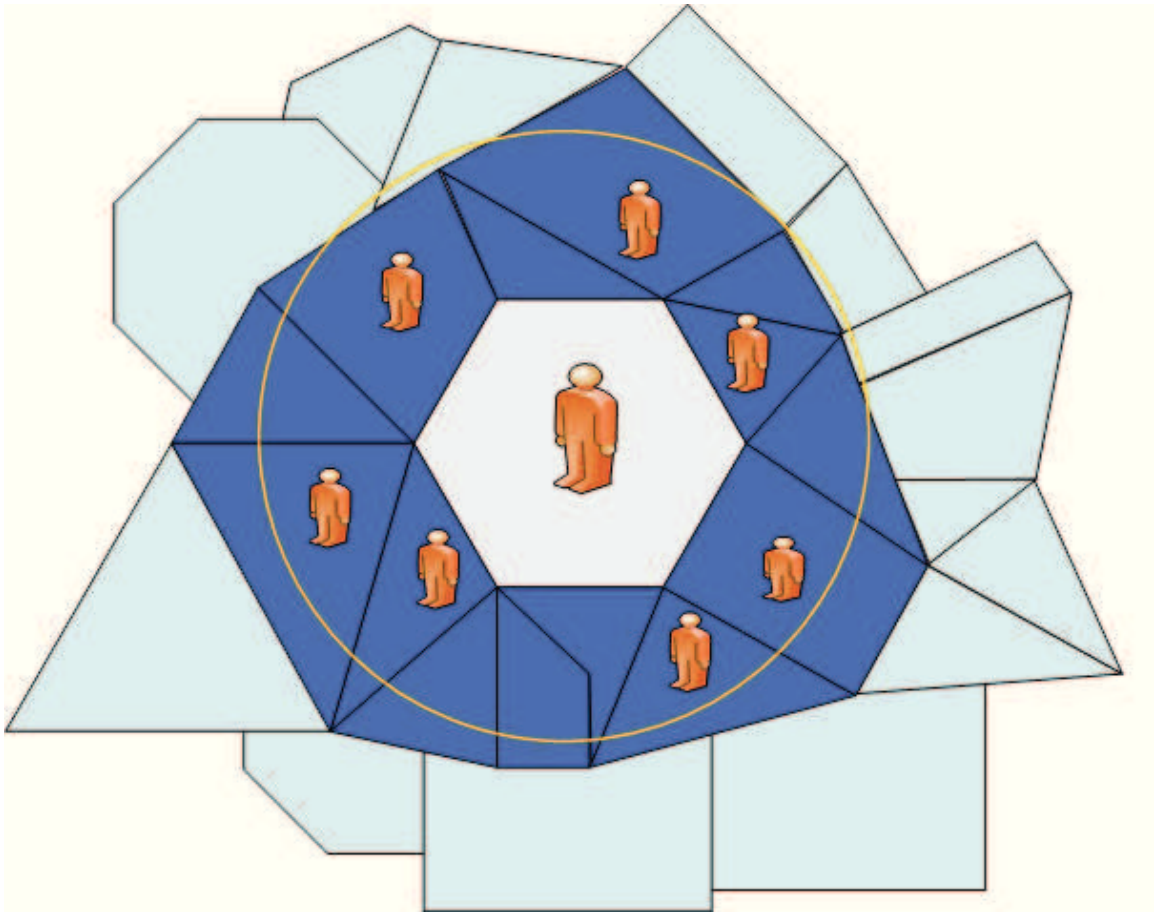


Figure 2.13: Voronoi diagram with aura overlapping enclosing neighbours (dark blue) and boundary neighbours (light blue) (adapted from [50])

Another approach is proposed based on peer-to-peer architecture and mathematical construct voronoi diagram [50]. The idea is that virtual plane world is partitioned into n non-overlapping regions of arbitrary size deterministically. It identifies neighbours as two types namely, *enclosing neighbours* which shares a common edge within a sensor range of a node/user and *boundary neighbours* are those overlapping AOI boundary as depicted in figure 2.13. The essence of this approach is that each user is aware of selected neighbours (instead of entire graph) and updates disseminated appropriately. As the user moves in the world, it re-computes the list of neighbours which can be an overhead if it moves constantly.

Some adaptive approaches to address IM problems in virtual worlds are reported in [89, 90, 91, 92]. This framework uses heuristic techniques to choose two forms of update

processing; *push-processing* sends an update message when a variable is written and *pull-processing* sends a request message when a variable is read (like ID query). It also extends to support range queries operations to access data efficiently. The idea is that it provides a light weight solution to address IM problems running on a global scale.

2.5 Summary

So far we have presented a survey of IM approaches used in several fields of distributed systems. The semantics of data access mechanism is largely dependent on the application and the focus here is on query semantics such as ID query and range query. IM approaches based on ID query systems such as ccNUMA (presented in section 2.2) proposed methods such as replication and migration to minimise the bandwidth and access latency. It also provides caching mechanisms to reduce the cost of accessing data. On the other hand, distributed systems such as peer-to-peer systems, multi-player online games, DVEs employing range query based systems in different forms such as auras, cells, range extents are presented. P2P systems build range query based systems over DHT and non-DHT overlays. The problem with using range queries over DHT overlays is that it poses a challenge for randomised and de-centralised usage of hash functions. Whereas, building systems with range queries over hash free P2P systems is simple and efficient to access data but it lacks randomness of data distribution across peers. In virtual environments, though scope of IM approaches is limited due to the fact that systems mostly employ flooding mechanism to update users, several approaches attempt to address the issue to improve scalability of the system.

2.6 Stage 2 : Parallel Simulation

This part of literature review covers a survey of problems within the context of parallel and distributed simulation. This section presents an overview of parallel simulation,

different notions and terminologies which will be used in the remainder of the thesis. Approaches within parallel and distributed simulation system attempt to distribute a simulation system across different hardware and software platforms that could be running at different geographical locations to achieve scalability and inter-operability. *Parallel and Discrete Event systems (PDES)* is one such system, which is a parallelised form of Discrete Event System (DES). There are some basic notions that needs to be explained before presenting mechanisms within PDES systems. This section is very important as it is the underlying system that is driven by and implemented in this thesis and the contents are adapted from [37].

A simulation model defines the physical system as events and states. Executing a simulation is an event, which updates the state and in therefore generates more events. There are two types of simulation models such as continuous and discrete simulations. Continuous simulation model has continuous state changes whereas the discrete model has state changes at discrete intervals. There are two ways of designing discrete models, *Time-Stepped* and *Event driven*. *Time-Stepped* models progress by changing the state from one time step to another, where the simulation is divided into an equal number of steps. *Event driven* models change the state of the system by discrete events. In this method, the state changes are triggered by events with a timestamp. Each processor has state variables, an event list and a global clock. An event is like a data record which requires space to store time, event type and related information. It is then added to the event list. Each processor has a event loop which takes the head of list and processes that event and reschedules new events, if any, to the event list. The simulation has start and end times.

The major factor of the simulation is to represent the time.

1. Real Time – the time in the real/physical system.

2. Logical Time – the time in the simulation model used as an representative of the real time.

3. Wall clock Time – the real time the simulation has been run so far.

The granularity of mapping real time to the simulation time varies on the simulation model. If the simulation model requires to run for million years then its not ideal to map every second in the simulation. The simulation model also gives opportunity to split tasks and run concurrently to speed up the simulation.

2.6.1 Discrete Event Simulation

Discrete Event Simulation (DES) models [36] are generally implemented as a queue of events each having a logical timestamp. A loop iterates over the queue, at each iteration dequeuing and executing the event with the lowest timestamp. The execution of an event may cause both transitions in the state of the system and also insert new events in to the queue to be executed in the future. However, the computational requirements of many simulation models far exceed the conventional sequential computer systems. This creates a need for **Parallel and Discrete Event Simulation (PDES)** solutions which allow a DES to harness the larger computational resources offered by parallel machines.

2.6.2 Parallel and Discrete Event Simulation

In PDES, a simulation model is decomposed into *Logical Processes (LPs)*, which are internally driven by a DES scheduling algorithm described in the previous section. In addition to populating the local schedule with new events, each LP can also receive external events from other LPs that are then scheduled locally with reference to a single global notion of logical time. The scheduling algorithm can create new local or external events to other LPs. Figure 2.14 shows an example of a PDES system.

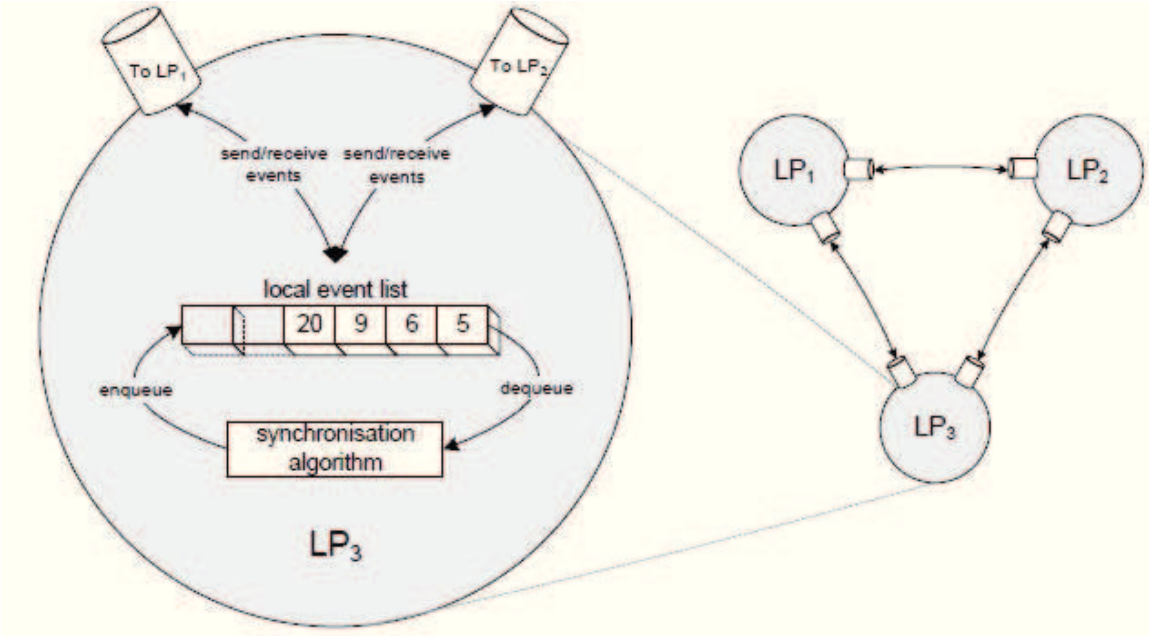


Figure 2.14: A Discrete Event Simulation Parallelised across three Logical Processes taken from [140]

Within such a PDES system, some synchronisation algorithm is required to make sure that events are processed in increasing timestamp order regardless of whether they are locally or externally generated. This condition is called the *Local Causality Constraint (LCC)*. In a parallel environment, it is difficult to ensure the LCC. The notion of logical time, time clocks and ordering of events in a distributed system have been introduced in Lamport [62]. The idea is that each process maintains a queue of pending request events waiting to enter critical section of the system. The ordering of the requests is based on the time notion introduced with the Lamport timeclock algorithm. The algorithm is based on clock consistency semantics which determines the ordering of events exchanged between processes with a simple mechanism. The clock can be used to determine the partial casual ordering of events between processes. However, the mechanism cannot be used to imply a causality relationship between processes.

There are several algorithms proposed to address the causality issue which is basically from two main families[120], one is the **Conservative approach**[21], which restricts the progress of an LP through logically time such that events are only processed if they are guaranteed not to violate the LCC and the other is the **Optimistic approach**[56] derived

from *Timewarp* concurrency algorithm, which does not restrict the progress of an LP but instead provides a mechanism for an LP to rollback processed events should an event arrive subsequently which would cause the LCC to be violated (such events are known as *stragglers*).

Both of these approaches have their advantages and disadvantages. Using conservative approach requires some knowledge about the simulation being modelled. In a sense, it makes assumptions about the simulation and predicts future events. On the other hand, optimistic approach allows simulation models progress at their own pace and time (logical) but requires saving the state of the simulation system to generate rollbacks. Memory required to store the shared state will increase exponentially if the simulation runs for a longer time (like for days). It is therefore clear that some mechanism is necessary to reclaim memory for the sustainability of the simulation system. To do this, one has to make sure that memory used by shared state will no longer be needed for further scrutiny (rollbacks, in this case). So, the system calculates a lower bound of timestamps of all LPs such that no rollbacks can occur. The lower bound time is called *GVT (Global Virtual Time)*. The idea is to take a snapshot of the entire state and decide a minimum time of all LPs called GVT such that memory used to store events that happened before GVT can be reclaimed and events after GVT will remain.

This can be done using a centralised approach to collect information across all LPs by blocking them during GVT computation. Due to the complexity and computation requirement, a distributed approach is required to compute lower bounds on all LPs. There are two problems[37], *simultaneous reporting problem* (where delays in network can give less accurate state of LPs) and *transient message problem* (messages sent from nodes might not have reached their destinations during GVT computation). So, a minimum of two iterations of GVT computation is performed to calculate accurately and consistently taking all these problems into account. There are different ways to design such algorithms and most common approaches are reported in [125, 86, 13, 22]. One such approach (adapted from Mattern's GVT algorithm[86]) implemented within PDES-MAS framework

is presented in detail in section 3.2.4 within chapter 3. In the following sections, a survey of adaptive synchronisation and load-balancing techniques proposed within PDES systems is presented.

2.6.3 Adaptive Synchronisation in PDES

Adaptive approaches takes advantages of both approaches (conservative and optimistic) or extends either of the approaches to support a simulation system. An adjustable moving time window is used in [80, 81] and a static bounded time window is used in [145] that are built on *Timewarp*[87] simulation system to process events. The idea is that a time window places restrictions on how far LPs can progress in the simulation. It creates a range window within which all events can be processed. Such approaches make sure all LPs progress in virtual time in a similar pace. When LPs progress at a similar pace, it reduces the risk of rollback scenarios. Apparently, choosing appropriate window size is a problem and is a typical trade-off.

Adaptive approaches mostly use information gathered either locally or globally to determine the outcome of processing an event in an LP. The following mechanisms use local state protocols which is the information available locally within a LP to determine the risk of processing an event. They may decide to block an event (in anticipation that this might create a future rollback scenario) or allow an event to be processed in a LP. [34] reports a technique based on CPU delay intervals. The idea is that it estimates a cost function which involves monitoring LVT progression in logical and real time. It also calculates the probabilities of a rollback cost and cost of blocking an event in terms of CPU cycles. Based on this approach, it either decides to process or block an event arriving in a LP. Similar to this approach, [85] proposed an algorithm Minimum Average Cost (MAC) which is based on a prediction technique to anticipate the arrival of a straggler event using probabilities that aims to minimise the cost of a rollback overhead and time delays in synchronisation.

Another approach uses a concept called *degree of optimism* which analyses the progress

of LPs in Local Virtual Time (LVT) and GVT to predict a rollback scenario [26]. The degree of optimism is a measure of how far optimistic a simulation model can progress (means processing an event without blocking). The real time measurements of such optimism is mapped against probability density functions to analyse and construct time windows for a LP. Similar approach using probabilities of events to asses the risk of processing them with or without blocking is reported in [33]. The information is gathered over time using access patterns. In [46], the communication channel specific information to and from other LPs is used to improve performance of the system. It reports performance improvements in some cases measured against standard optimistic and conservative mechanisms. In [108], time windows for LPs is calculated based on number of rollbacks, anti-messages, rollback length and committed events to ascertain whether to process an event or not. The window size is calculated dynamically and it reports better performance than standard Time Warp system.

Slightly different approach reported in [109] used uncommitted events and progress in GVT to calculate window size (in contrast to using rollbacks). The window size is adaptable and the upper bound of time window is changed based on the measurements taken periodically. Another approach proposes a system called a shock resistant TimeWarp simulation system[35]. Instead of reacting to the changes in the environment termed as shocks, it predicts well in advance to outperform reactive mechanisms.

On the contrary, the following approaches use well gathered information (global) across the simulation system to improve the performance. The problem of using local state is limited information and may not be accurate. Collecting global information of the entire shared state of all LPs will give clear idea of the simulation progress. But it increases the overhead of computing such information. One such approach is reported in [27, 28] which uses an adaptive memory management protocol to restrict the optimism of *timewarp* by varying the amount of memory available. It monitors the performance of the system periodically to ascertain the availability of memory for events to be processed.

Another scheme called Near Perfect State Information (NPSI) [132] restricts optimism

using a protocol called *error potential*. It controls the optimism of LPs executing an event. However, this approach is restricted to Shared Memory Multi-Processors architecture as NPSI is built on that assumption. Elastic Time Algorithm (ETA) is an example of such an approach which defines error potential to be the time difference between the next event on a LP and GVT[132]. The degree of optimism is then calculated to either block or process an event in a LP. Another approach proposed in [139] distinguishes between slower and faster LPs (in terms of progress in LVT) and determine the events to be processed on each LP. It restricts the progress of faster LPs and allows slower LPs to catch them.

Similar adaptive approaches proposed within PDES-MAS framework that uses access patterns of agents to predict rollback scenarios based on time windows (global state) and probabilities (local state) is reported in [72, 73, 65, 64, 66, 74, 75]. Though results show better performance in reducing rollbacks and simulation time, it has not been tested with a large scale distributed environment set-ups.

2.6.4 Load Balancing in PDES

The problem of load balancing is orthogonal Large scale PDES systems require large scale data structures to store shared state across LPs. As simulation progresses over time, a possibility to overload a LP/LPs will potentially arise. Balancing the load across LPs is a vital requirement for scalability and operability of the simulation system. There are several ways to balance the load such as migrating the shared state across LPs or creating new LPs to share an overloaded LP and merging or deleting light-weight (or no weight) LPs in the simulation system. If the simulation system is quite simple, some static approaches can be used. The distribution of shared state can be decided in prior using the knowledge of the system, number of LPs involved and bandwidth (number of messages exchanged between LPs) of the system. Several approaches proposed based on such mechanisms are reported in [102, 17, 58]. On the contrary, some used more dynamic or adaptive mechanisms to distribute shared state. The decision to partition or merge state across LPs can be done in two ways: centralised and distributed.

Centralised approach assigns one LP to gather information across the system and decide distribution strategy across them. The idea is that assigned (or nominated) LP gathers statistics across LPs to initiate load balancing algorithm. Some centralised based approaches are reported in[18, 43, 127, 6]. The obvious issue is that overhead in computing global state information and also overloading the nominated LP itself. On the other hand, distributed approaches use local information gathered over time to decide distribution are reported in [101, 118, 126, 149, 143].

On the other hand, adaptive approaches gather both local and global information periodically to ascertain distribution. One such adaptive load management mechanism proposed within PDES-MAS system is reported in [105], which uses access patterns of LPs to migrate a portion of shared state closer to the accessing LPs. It uses several threshold parameters to avoid thrashing the system with migration. The experimental results show that the cost of accessing variables in a distributed environment and the bandwidth of the system is reduced significantly.

2.7 Stage 2 : Agent Based Modelling

This section presents an overview of Multi-Agent Systems (MAS) and how simulation, in particular PDES techniques, helps agent designers to understand and develop their systems. Common applications of agent based systems include control of mobile robots, computer games, telecommunications and military applications. 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 via message passing [150]. The environment of an agent is a part of the simulation world or representation of computational system. Agents inhabit within this world have either disjointed or partially overlapping environments with each other. Different definitions of an agent exist in agent based modelling community and a survey of such definitions is presented in [123]. In general, an agent has the ability to co-exist in an environment sensing its surroundings and

act upon the environment based on its perception. Another term commonly used is situated agent which can sense and perceive in a dynamic environment. An Embodied agent is an intelligence agent which has a physical presence in the environment (e.g.. humans in virtual world). A mobile agent is one which moves within the environment. Multi-Agent Systems (MAS)[57] are often extremely complex and it can be difficult to formally verify their properties. As a result, design and implementation remains largely experimental, and experimental approaches are likely to remain important for the foreseeable future. In this context, simulation [79] has a key role to play in the development of agent-based systems, allowing the agent designer to learn more about the behaviour of a system or to investigate the implications of alternative architectures and the agent researcher to probe the relationships between agent architectures, environments and behaviour. For this work, the agents are perceived to be embodied, situated and mobile existing in large dynamic and non-deterministic environment.

There are various simulation toolkits available for MAS such as MASON [82], RePAST[103]. A survey of several toolkits is reported in [128, 42, 115, 140] and how each toolkit behaves in terms of system architecture, performance and agent's behaviour etc. Although real MAS software frameworks and the real-world systems that MAS simulations represent are continuous systems, MAS simulations are typically implemented as **Discrete Systems**. Since a MAS is a complex system, small perturbations in event sequences or agent decisions early on in the play-out of a simulation can radically change the results. Therefore, a de-centralised, event-driven distributed simulation is particularly suitable for inherent asynchronous parallel systems such as agent based system.

Several approaches have been proposed using parallel DES paradigms for non-traditional DES systems such as MAS[79]. Some approaches use time stepped DES techniques such as Genism and Dgenism [4], HLA_Agent [71, 69, 70] and HLA Jade [147]. Others used event driven DES techniques such as HLA_Repast [88], SPADES [121] and CHARON [3].

2.7.1 Interest Management in DS-MAS

Over the years, several approaches used PDES techniques to simulate intelligent and autonomous agents inhibiting within a complex, dynamic and non-deterministic environment of MAS. Accessing data consistently and in a time ordered fashion using PDES techniques is always a challenge in such systems. As these systems become larger and more complex, addressing the issue is a vital requirement of the scalability of the system. In Genism and Dgenism[4], a perception of an agent is its visibility range (similar to a range query) in the physical environment. The perception returns with all objects and it is the responsibility of agent manager to filter required information. Though the system provides a sophisticated perception to an agent in a distributed manner, it attributes to an overhead of data transmission for large number of agents. Such overhead is avoided with restrictions imposed on frequency of updates sent between the environment and agent managers with the knowledge of agent's current location and sphere of interest.

In HLA_Agent [71, 69, 70], each federate (simulating node) maintains its own local information of objects that shares data frequently. Each federate updates other by *publishing* interested attributes and also *subscribes* to attributes which it likes to receive updates in the future. Such *publish-subscribe* method is declared at the start of the simulation using DDM (Data Distribution Management) services. It assigns multicast groups statically based on prior knowledge of federates connection patterns. Such schemes may not be useful for non-deterministic agent based modelling. This is because agent based systems are assumed to dynamic, non-deterministic in nature and behaviour. Assigning federation and multicast grouping would limit the number of federates in the simulation and their movements within the system. HLA Jade [147] uses a similar HLA-DDM management service to register their interested data. However, the difference is that the environment is split into cells and at any point of time, a federate is confined to a cell. Its interest is restricted to its immediate cells (i.e., surrounding eight cells and its own cell), typically its sensor range. Whenever there is a match or an overlap between subscribe and update region, updates are sent automatically using HLA-DDM multicast services.

Obviously, the disadvantage is that mechanisms required to calculate matching regions at every time step will be an overhead.

The systems mentioned so far use time stepped DES techniques which pre-determines the granularity of time steps of each agent in the simulation. However, the following approaches use discrete event techniques allowing agents or simulating nodes to progress at their own pace. Such approaches require a rollback mechanism to correct the ordering of events (if events are processed out of order). HLA_RePast[88] is a discrete event system but uses conservative mechanism to synchronise events avoiding any rollback scenarios. In a way, it is similar to time stepped events except that each agent's progress is used to predict their future events individually. It also uses a similar scheme of publish and subscribe mechanism to update changes in the state of objects and attributes. It has the advantage of simulating autonomous agents more precisely and allow simulation nodes to progress at their own pace. It uses *remote invocation* methods to keep changes up-to-date. Similar conservative approach is proposed in SPADES [121] that provides a centralised approach for agent communication. Every update from an agent is sent to all communication servers and then to all agents, which is similar to a broadcast communication. Agent's region of interest is irrelevant in such systems. Such mechanism cannot be suitable for large scale simulation systems such as MAS.

In a summary, systems mentioned above all used DES techniques (either time step or discrete event) to simulate MAS. The advantage of using discrete event techniques is that it allows agents or simulating nodes to progress at their own pace. This clearly simulates the autonomous and intelligent agent based models. However, none of these approaches completely exploit autonomous agent behaviour and access data efficiently. The advantage of using discrete event techniques is almost nullified by using conservative synchronisation mechanism to restrict the progress of the agents. Such mechanisms require complete knowledge and predictable agent behaviours in the system.

2.8 Data access mechanisms in PDES systems

Systems based on PDES pose a challenge to build a large scale simulation system such as Multi-Agent Systems (MAS) in terms of scalability and inter-operability as reported in section 2.6. Of several issues within such system, addressing data access mechanism is a significant requirement for the scalability of the system. The unique problem with involving PDES techniques is the notion of logical time and its restriction on ordering of events. Whereas such constraints are not applicable for data access mechanisms and their approaches in different fields such as peer-to-peer systems (P2P), DVEs, MMOGs and distributed and shared memory systems. Though ordering of events is important in some systems, they can still mask these inconsistencies and produce acceptable results. Approaches presented in section 2.7, though intend to address such issues it fails with the requirement of global knowledge to impose restrictions on the progress of simulation system (to avoid rollback scenarios).

In this context, this thesis presents a notion of time synchronised range queries and updates and state migration algorithms implemented within PDES-MAS (Parallel and Discrete Event Simulations for Multi-Agent Systems) framework. PDES-MAS provides a framework to distribute and run parallel simulation of Multi-Agents. It employs optimistic synchronisation mechanism to make sure that the events are processed in timestamped order according to LCC. Such mechanism allows simulation nodes to progress at their own pace and invokes rollbacks for processed events incase LCC is violated. This property is important to verify and analyse MAS as even small changes in the order of processing events can change the outcome of the system. An overview of PDES-MAS architecture will be presented in chapter 3 and synchronised range query and state migration algorithms implemented within this system will be detailed in chapters 4 and 5.

CHAPTER 3

PDES-MAS

This thesis presents a notion of synchronised range queries within the PDES-MAS framework. The algorithms are built based on the assumption and philosophy of PDES-MAS framework. So, it is vital to present an overview and architecture of PDES-MAS framework. PDES-MAS is a distributed simulation framework specifically designed to support large scale MAS models. Agent based models are complex, dynamic and non-deterministic in nature and behaviour and so often distributed simulation plays an important role to understand and design such models. There is always a challenge to execute MAS models on parallel computer platforms. Such problems are aggravated as the scale of MAS increases. A survey of approaches proposed towards that direction is reported in [140]. One such approach is to partition the shared state of simulation model and distribute across available computing resources. In this context, PDES-MAS is built based on distributed shared memory structure whereby publicly accessible variables are modeled as Shared State Variables (SSVs) and agents access them using reads (queries) and writes. A detailed overview of PDES-MAS framework is presented in the following sections.

3.1 Modeling MAS in PDES-MAS

PDES-MAS is implemented adopting PDES paradigm to distribute and run parallel simulation of MAS. A Discrete System approach in which the simulation progresses by atomic

state changes at discrete intervals, allows such simulations to produce more robust and repeatable results[37]. Following PDES approach, the simulation model is divided into Logical Processes (LPs) each maintaining a disjointed portion of a state space of the simulation system. State changes are modeled as timestamped events in the simulation. Within an LP, there are two types of events namely, internal events which have a casual impact on the interval variables of the LP and the external events which may have impact on state of other LPs. External events are modeled as messages with timestamps. In PDES-MAS framework, the agents and their environment are modeled as **ALPs (Agent Logical Processes)** and **CLPs (Communication Logical Processes)**. ALPs imitate the actions of agents by sensing (queries) and updating (writes) the environment and also maintain the internal state of agents. CLPs model the shared (public) state of the system in which ALPs can operate on the state. The design details presented in the sections from 3.1.1 to 3.1.5 are adapted from [79, 68, 67].

3.1.1 Concept of Shared State

Agents in a simulation can access the shared state of the system. The shared state is otherwise called public state on which more than one agent can operate on concurrently. Multi-Agent systems (MAS) are highly dynamic and each agent can operate on a different portion of the state at different periods of time. The shared state is modeled as a list of variables. These variables are termed as **Shared State Variables (SSVs)**. Agents can either access (query or write) and modify (add or delete) these SSVs. The assumption is that each agent goes through '*sense-think-act*' cycle. As the simulation progresses, each agent can go through several of these cycles. The essential idea of this cycle is that each agent senses the environment before they act on a shared variable. Several agents can access or modify the same variable at the same time. An action may be confined to just pre-condition check or pre-condition, action and post-condition (usually outcome of the action) or just post-condition check within a cycle to make sure that agent's action is performed on the system. For example, if an agent $A1$ queries the location of a variable

$V1$ as (x, y) and sends a write to move the variable to location $(x1, y1)$ based on the read value. But in the meantime, some other agent $A2$ has changed this variable location to $(x3, y3)$. Now, the write action by the agent $A1$ based on its assumption of the value of variable $V1$ is not valid. But there are several ways to handle such scenarios and it depends on the assumptions of the simulation system.

3.1.2 Modeling Shared State

PDES-MAS framework models shared state of the simulation in terms of objects and attributes. Each SSV class represents an attribute of an object. There can be many attributes for an object i.e., an object representing a location (x, y) has two attributes namely x and y. Each action (query/write/add/delete) operates on a SSV class. The shared state of the simulation, thus, has a collection of SSV classes. The shared state is logically a vector of tuples of the form

$$< object\_type, object\_id, attr\_type, attr\_id, value, timestamp > \tag{3.1}$$

As the simulation progresses with time, more tuples are created as the SSVs are updated with new values. New SSVs are also created in this process. The entire shared sate of the simulation has to be maintained synchronously at any given time. It also depends on the synchronisation mechanism implemented on the framework, especially optimistic synchronisation. If the synchronisation mechanism is optimistic, then all tuples greater than GVT (Global Virtual Time) time has to be maintained. Each ALP can perform operations such as query, write, add and delete on the shared state of the simulation. CLP maintains the shared state of the simulation which receives these operations. Agents's actions are transformed as operations implemented in this framework. There are four operations namely,

- Queries - There are two types of queries such as ID query and Range Query. An ID query senses a specific SSV at logical time whereas a Range Query senses a set of

particular SSV class with values that match a specified range.

- Writes - Updates a SSV with a new value and timestamp.
- Add - Creates a new SSV with a value and start time.
- Delete - Deletes the SSV after the specified timestamp.

As agents go through 'sense-think-act' cycle, they express their actions transformed into operations on the shared state. Each agent senses the environment, process the information and decide to act on an object in the environment. The sensing action is modeled as reads (queries) in PDES-MAS framework. Now, the agent can decide to query a particular object or a list of objects within its sensor range. ID query can access the information on a particular object and Range Query can return with a list of objects within the sensor range or range window. Acting on an object is modeled as a write to an object with a new value. It is termed as ID write as it is not possible to update multiple objects at the same time. An agent or environment can decide to create a new object which is modeled as an add operation and removing the object from the environment is modeled as a delete operation.

3.1.3 Synchronisation

PDES-MAS framework uses optimistic synchronisation mechanism to make sure that the events adhere to the Local Causality Constraint (LCC). This mechanism allows events to arrive out of order in logical timestamp but rolls back to correct the state of the simulation. So, the framework uses a state saving mechanism to correctly rollback an operation. It introduces mechanisms to define an order of processing events and resolving the conflicts on processing concurrent operations. Following the agent's cycle, the correct order of events with the same logical timestamp is as follows,

- Queries

- Adds
- Deletes
- Writes

The ordering mechanism allows agents to access or modify the shared variables in a time synchronised manner. It means that an agent cannot act (add/delete/write) before sensing (query) the environment. It allows agents to add and delete a variable at the same time. Also, the agent can write to a variable after reading it at the same time. A rollback is triggered if any of the operations arrive out of order on a variable. A conflict in the validity of a value of variable at time t (logical) arises, when two writes from two different agents arrive at time t for the same variable. Such conflicts of writes can be resolved in many ways. In this framework, a simple comparison of IDs (where a unique ID is assigned to each agent) of agents decides the validity of a write (here, write arrived with highest agent ID is taken as valid). However, it is not easy to operate on vector of tuples to ascertain this mechanism. So, this framework has a data structure to store the state of the simulation as a list of Shared State Variables (SSVs). Each SSV has a list of *Write Periods* where each write period represents the validity of a value with a start s and end time t of a period $(s, t]$. Each write period also maintains a list of ID queries arrived in that period $(s, t]$. A snapshot of a write period is depicted in figure 3.1. A straggler write at time t' in this case can modify an existing write period or split it. Rollbacks are triggered to ALPs that issued premature reads (queries) that have timestamp $> t'$ using the write period.

3.1.4 Distributing Shared State

Distributing shared state of the simulation is very important and required for the longer run of the large scale simulations such MAS. Maintaining the entire shared state of the simulation in a CLP will be a bottleneck. For MAS it is often difficult to predict and distribute shared state using conventional PDES techniques. It is not possible to determine

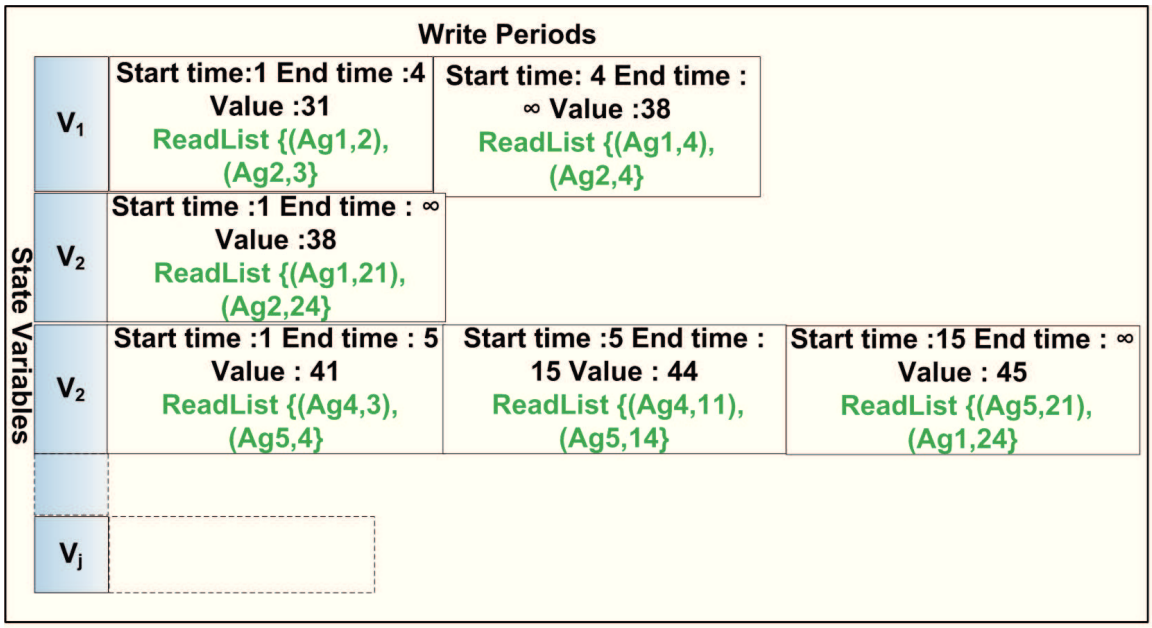


Figure 3.1: Write Periods Data Structure (taken from [67]).

the topology in prior. PDES-MAS presents a notion called *Sphere of Influence (SoI)* to distribute the shared state of the simulation. The essence of the notion is that a cluster of CLPs serve requests from ALPs with each CLP maintaining a portion/subset of the shared state variables and interactions with the ALPs. This has to be dynamic and light-weight solution to make sure that the load is balanced among CLPs. There are different ways to achieve dynamism (discussed in detail in chapter 5), of which this framework has a fixed tree of CLPs where shared state of the simulation is distributed among them. More comprehensive and adaptive approaches are discussed in [105]. The idea is that shared state of the simulation in the CLP tree should reflect the SoI of the ALP interacting with them. This relies on the assumption that SoI does not change drastically over short time periods of the simulation. An analysis on SoI mechanism over different agent simulations has been presented in [79, 72]. The set of LPs is thus arranged in to a sparsely connected graph and, more specifically, a tree of which ALPs are connected as leaf nodes and CLPs as the root and intermediate nodes as depicted in figure 3.2.

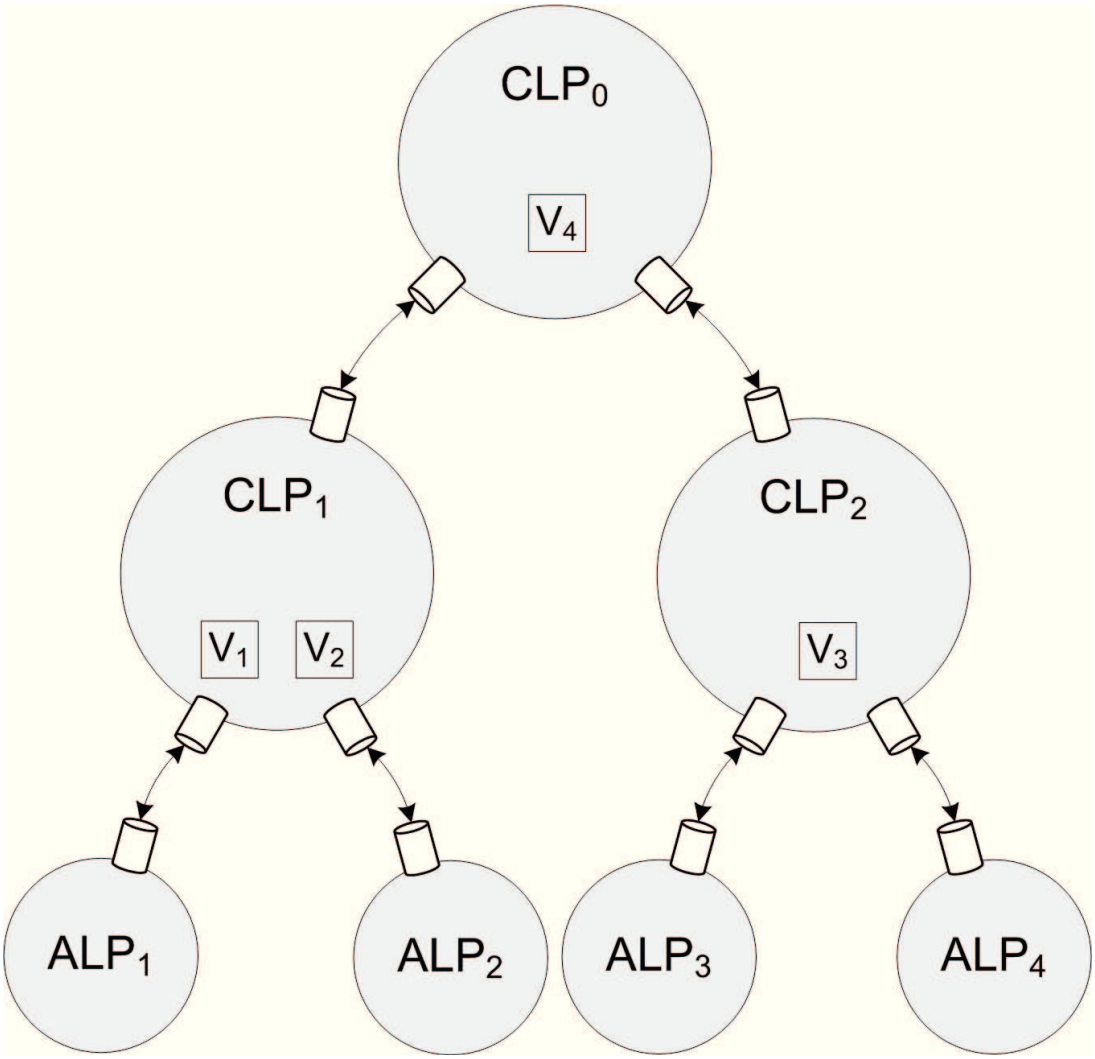


Figure 3.2: A tree of 3 CLPs and 4 ALPs.

3.1.5 Operations on Shared State

The framework supports operations such as ID query and Write on a SSV, rollback on a ALP (in case of incorrect ordering of events) and GVT (garbage collection of shared state). Each CLP has four ports; an internal port H (Here) and three external ports U, L and R (Up, Left and Right) through which it is connected to neighbouring LPs. A snapshot of a CLP structure with external ports is depicted in figure 3.3. Each CLP has a list of SSVs and their write periods evolved over time. At this stage, lets assume that each CLP is also aware of locations of SSVs and ALPs in the tree. When an ALP issues a ID query for a SSV, it forwards the request to its parent CLP (the CLP it is

attached). If the parent CLP maintains that SSV, it returns with a response otherwise it forwards through the port containing that SSV. The ID query is thus forwarded to the CLP holding the SSV and follows the same route to return with a response. When two or more agents write on a SSV at the same logical time, the agent with the largest ID will have the priority to create or update the write period and the rest will be neglected.

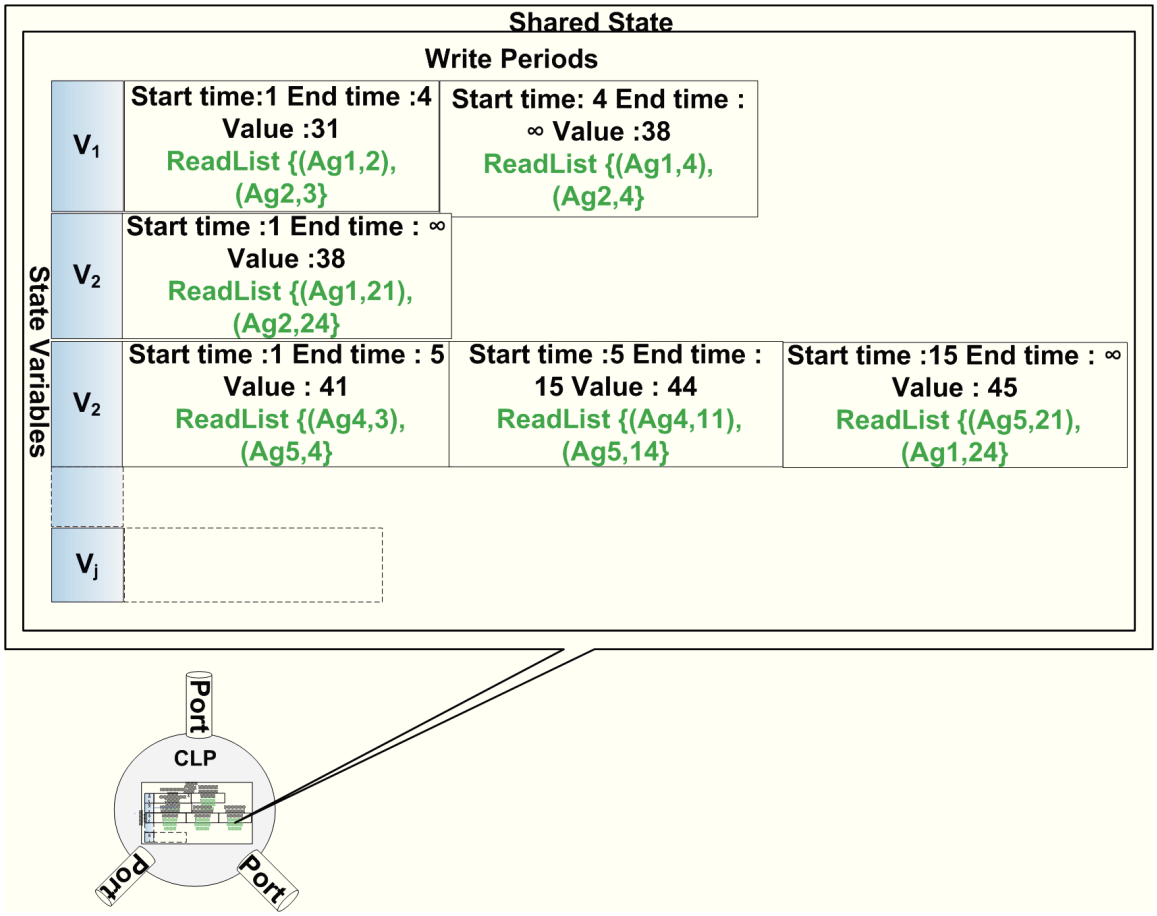


Figure 3.3: CLP, ports and shared state (taken from [67]).

Each CLP is also responsible for generating rollbacks to ALPs, when a straggler write arrives for a SSV thus invalidating previous queries or writes. The rollback for an ALP is forwarded through the ports that lead to the parent CLP to which it is connected. ALP on receiving a rollback with time t' rolls back its LVT to t' thereby sending anti-messages for all events sent before with time $> t'$. The anti-messages make sure that all actions executed with time greater than rollback time is removed from the shared state of the simulation. Each ALP maintains an event list to execute such mechanism.

In summary, MAS simulations are modeled as distributed, de-centralised and discrete event systems exploiting the inherent parallelism of agents. The shared (public) state of the simulation is modeled as vector of SSVs each with a list of write periods storing values at each write period. ALPs model the agents actions in the simulation and CLPs maintain the shared state of the simulation storing the interactions of agents with the SSVs. The framework adopts optimistic synchronisation mechanism to make sure that the events adhere to Local Causality Constraints (LCC). It is designed to support both access (query/write) and modify (add/delete) operations. Rollbacks are generated to correct the ordering of events processed in the simulation. The ordering and conflict resolution of concurrent operations are as defined in section 3.1.3. For a large scale simulations such as MAS, a very large scale data structures are required to maintain the shared state of the system. So, a cluster of CLPs is used where each CLP maintains a portion of disjointed shared state of the simulation.

3.2 Architecture of PDES-MAS

In this section, we present an overview of modules and their functionalities implemented within PDES-MAS framework. The architecture of PDES-MAS framework is presented in figure 3.4. On the basis of PDES paradigm, the framework models two types of LPs: An **ALP (Agent Logical Process)** simulating agent's behaviour and a **CLP (Communication Logical Process)** maintaining a portion of shared state of the simulation. Modules within each type of LP are depicted in the figure 3.4.

3.2.1 Initialisation

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

1. A list of SSV IDs, their data types and initial values. For example, a SSV repre-

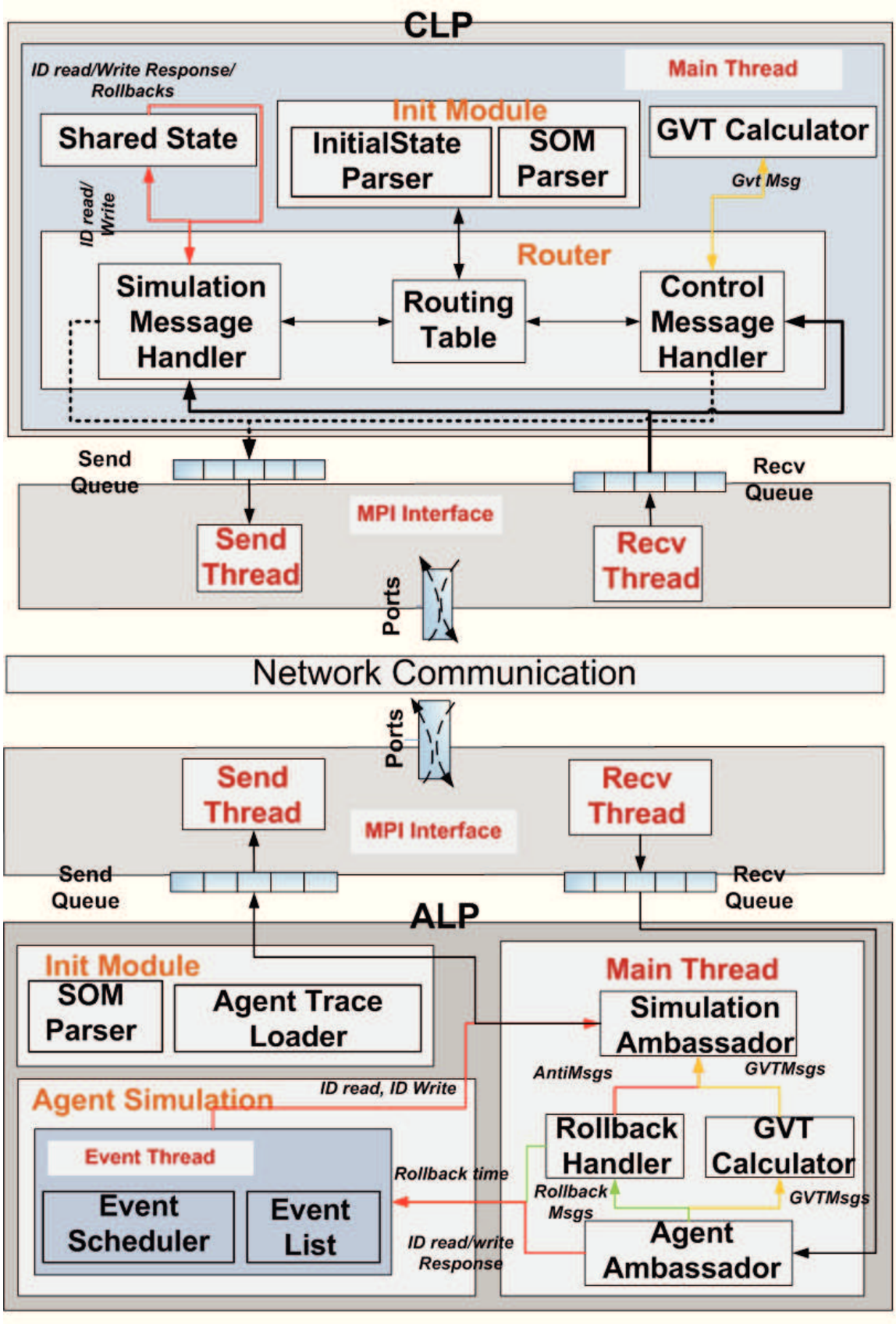


Figure 3.4: PDES-MAS architecture.

senting its *size* can be modeled as an *INTEGER* data type.

2. A list of simulation interactions in the framework. For example, an agent moving an object in the simulation is modeled as a *WRITE MESSAGE* with a new value to the object.

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

4. A list of event traces representing agents actions in the simulation. The format for each type of event supported in this framework is presented below.

InitialStateParser module loads with a list of SSVs IDs and their initial values. A SSV ID is associated with a unique id format *(ObjectInstanceId, VariableClassId)*. *ObjectInstanceId* represents a type of an object in the simulation (e.g.. location) and *VariableClassId* represents an attribute type of the object (e.g.. x/y position). All these informations are stored in a XML file, which is then parsed and loaded into a CLP. **SOM Parser (Simulation Object Model)** module, notion commonly used in HLA [54] technology, prescribes with types of simulation objects, attributes and interactions with the simulation model. The interactions (messages) in this framework are of two types namely, *Simulation Messages* (ID query, ID write, Rollback, Anti-Messages (query/write) and Response messages) and *Control Messages* (GVT and Load balancing Messages).

ALPs are not real agent behavioural models but rather simulate actions using pre-loaded traces generated from an agent simulation model. It does not hold any private variables. Similar to CLPs, the SOM parser module in an ALP loads simulation objects and attribute types and interactions with the framework. But ALPs would not be aware of the locations of SSVs in the CLP tree and simply forward messages to the CLP tree. Each ALP loads a set of events (actions in the simulation) using *Agent Trace Loader* module. An ALP can hold a list of actions of an agent in the simulation model. Any agent simulation model which follows '*sense-think-act*' cycle will fit this framework as long as it adheres to the following action message (traces in this implementation) formats specified below.

- ID Query:

$$< agent\_id, read, object\_instance\_id, variable\_class\_id, timestamp > \quad (3.2)$$

The labels are self-explanatory and a simple example of a ID query is given below

$$< 10, read, 1, 2, 100 > \quad (3.3)$$

where 10 is agent_id, action is read (query), SSV id is $(1, 2)$ (object_instance_id is 1, and variable_class_id is 2) and 100 is *timestamp* (logical time). The trace format for ID write is specified below which bears a similar format except for value (new value for the SSV id).

- ID Write:

$$< agent\_id, write, object\_instance\_id, variable\_class\_id, value, timestamp > \quad (3.4)$$

In case of ID Write, SSV id is accompanied with a value (presumably a new value).

The events are generated concurrently from ALPs that and are forwarded to the CLP tree. At this stage, the framework supports access to variables of primitive data types such as INTEGER, FLOAT, STRING, LONG, and DOUBLE. As mentioned above, the framework is initialised with IDs, locations and types of SSVs in the CLP tree and event traces of agents in the simulation. This requires frequent I/O access, parsing and construction of several tables within a LP.

To provide an indication of the relative time spent at the initialisation phase, the investigation is carried out on an allocated 8 cluster nodes. Each node has 4 cores of

$2GB$ memory and in total 32 cores are used for our experimentation. An ideal setup of 1 process (ALP/CLP) per core takes on average 0.3 msec to initialise a SSV in a CLP and 0.05 msec to load a trace in an ALP. With this setup, it takes 235 msec to initialise both ALPs (450 traces/ALP) and CLPs (680 SSVs/CLP) in the framework.

3.2.2 Message Handling

This section presents mechanisms to handle messages arriving asynchronously at different ports within a CLP. Each CLP has four possible ports of which an internal port H('HERE') and three external ports U, L, R (UP, LEFT and RIGHT) connected to neighbouring LPs. A CLP can receive two types of messages through these external ports such as *Simulation messages and Control messages*. Within a CLP, 5 different queues handle messages being sent and received through different ports. Two queues (*Send and Receive Load*) are dedicated to send and receive load balancing Messages with higher priority, a *Send Control Queue* to send control messages and two normal queues (*Send and Receive*) to send and receive simulation messages with the least priority. This ordering is used within a CLP to process messages arriving at different ports.

The framework uses MPI (a standard MPI-2) libraries to establish communication between LPs (for more details, see section 3.2.7). It provides an interface through which messages are sent and received from LPs asynchronously. *MPI Interface* module implements two threads, namely a *Send* thread and a *Receive* thread, to send and receive messages from other LPs. Two threads work asynchronously to enqueue and dequeue a message from the queues (based on the priorities mentioned above). Messages are processed synchronously within a CLP using a *Main thread*. Semaphores are used to communicate between *Main thread* and *MPI Interface* threads to process messages synchronously.

For example, if a ID query for a SSV arrives at a port, the *MPI Interface* queues the message to the Receive (normal) queue and signals *Main thread*. The *Main thread* within a CLP processes the message from the receive queue. The message is then forwarded to the destination port. If the SSV is maintained locally (port - H), then *Shared State*

module (a list of SSVs and their write periods, see section 3.4) processes the request and constructs a response message. The response message is then buffered into the Send (normal) queue and signals *MPI Interface* module to send the message. Main thread will not dequeue the next message until the existing message is processed and queued into the send queue. This synchronisation mechanism ensures that *shared state* module is accessible for only one request at a time.

3.2.3 Agent Message Handler

This section presents message handling mechanisms within an ALP. Event flow structure inside an ALP is depicted in figure 3.5.

Within an ALP the communication with the CLP tree is established via two structures:

1. *Simulation Ambassador*, which transforms an event from an ALP into a message format and forwards to the parent CLP. Since *SOM parser* is pre-loaded with the interaction (message) formats, the transformation is quite straight forward.

2. *Agent Ambassador*, on the other hand, takes care of handing over the response messages to appropriate modules.

Within an ALP, an *Agent Simulation* module models the life cycle of an agent. It maintains an event list and an event scheduler to update its LVT (Local Virtual Time). A LVT of an agent represents the latest timestamped event sent from that agent. An *event thread* runs separately to maintain and schedule events from the event lists. An *event thread* dequeues an event of an agent, update agent's LVT, converts it to a message using *Simulation Ambassador* and enqueues into the send queue. The event scheduler waits for a response (in case of ID query/write). When an ALP receives a response message, the *Main thread* signals the waiting event scheduler to yield and process the next event. The scheduler does not wait for other events such as GVT. Only a rollback message can interrupt this cycle and resetting the scheduler of the agent to the rollback time.

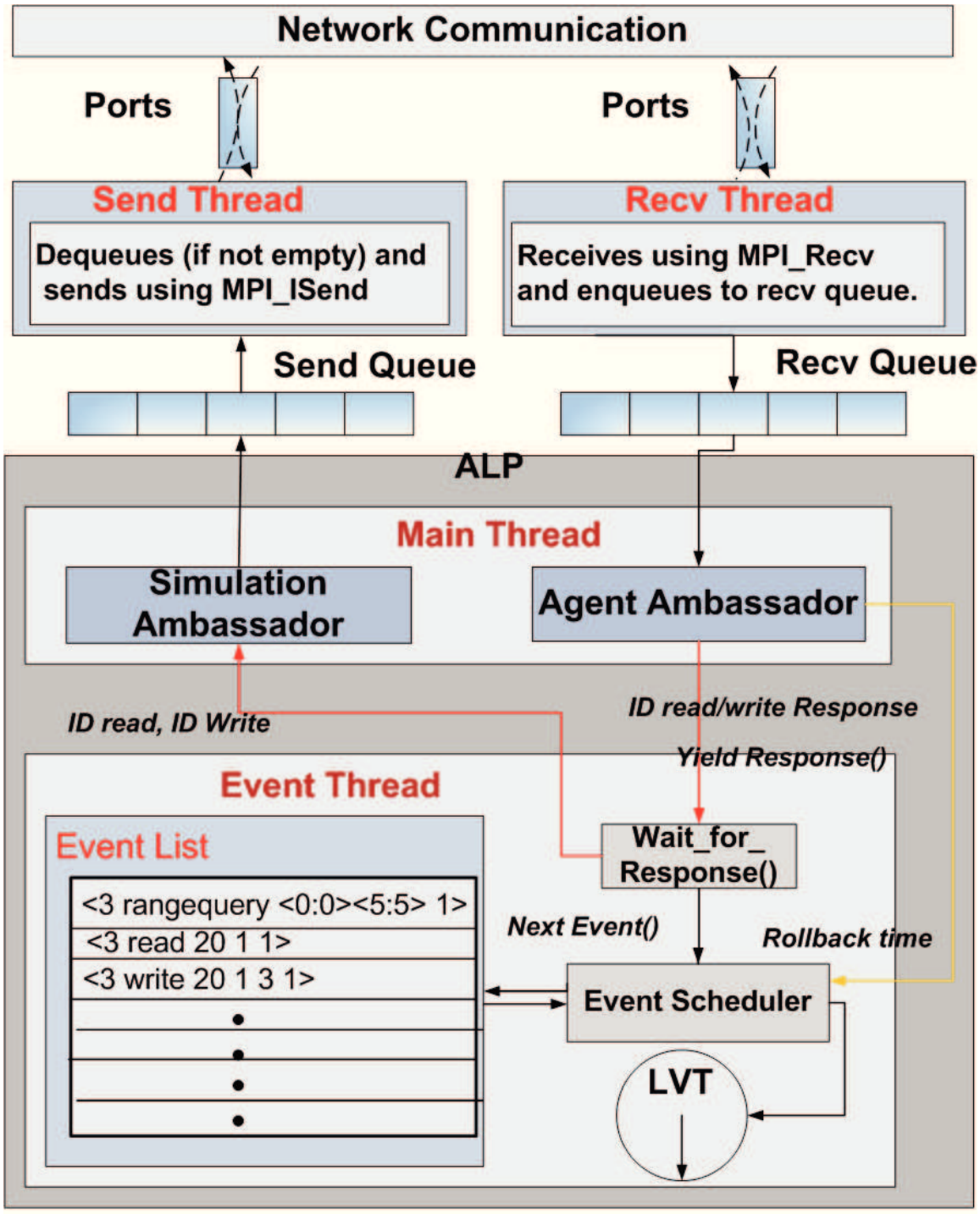


Figure 3.5: Event flow structure inside an ALP.

3.2.4 Synchronisation

PDES-MAS employs optimistic synchronisation mechanism to synchronise the events on a SSV. Rollbacks are triggered when events are processed out of order on a SSV. Typically,

a rollback scenario would be an arrival of a straggler write (in real time) arriving with (logical) time t′ on a SSV invalidating premature ID queries (premature, in the sense, reads arrived earlier in real time with logical time > straggler write) issued for that SSV at time > t′. To enable such mechanism, each SSV is associated with a list of of *Write Periods* representing the values taken by the variable at different logical time periods of the simulation as illustrated in figure 3.1.

A rollback message is generated to agents that issued premature reads (ID queries). *Rollback Handler* within an ALP, on receiving a rollback message for an agent, say, at time t′, checks its validity (whether t′ < LVT of the agent, otherwise rollback ignored). On receiving a valid rollback, handler holds the event scheduler from processing any event, sends anti-messages for all events with time > t′ and updates the scheduler with the rollback time t′ for that agent.

3.2.5 Garbage Collection

The optimistic PDES-MAS system keeps the history of state changes in the simulation. The memory consumed by these states could affect the progress of the simulation on a longer run. This problem is tackled by reclaiming memory whenever any processor is on the verge of running out of memory. But it has to determine the removable states without disturbing the existing simulation. A common way to achieve that is to get a snapshot of the system and determine the removable states. Several algorithms proposed for this purpose are reported in [125, 86, 13, 22]. This framework implements an adaptation of Mattern's GVT algorithm[86].

As mentioned before, in a distributed system, each logical process (here, ALP) has their own notion of local virtual time (LVT) like local clock. To determine the removable states, we need a horizon (GVT) like global clock time or server time, in which all states before that could be reclaimed. But in a dynamic system of processes sending messages randomly, the snapshot should somehow make sure it has got enough stable information to determine the horizon. The algorithm defines a cut at time t, in which all messages

sent before t as PAST and after t as FUTURE. This cut determines the PAST states and could be removed safely. But defining this cut in a real time is not easy. See figure 3.6, messages in PAST from process LP3 reaches LP1 in the FUTURE and so on. In a distributed environment there are no guarantees for the arrival order of messages, so one has to take care of transient messages.

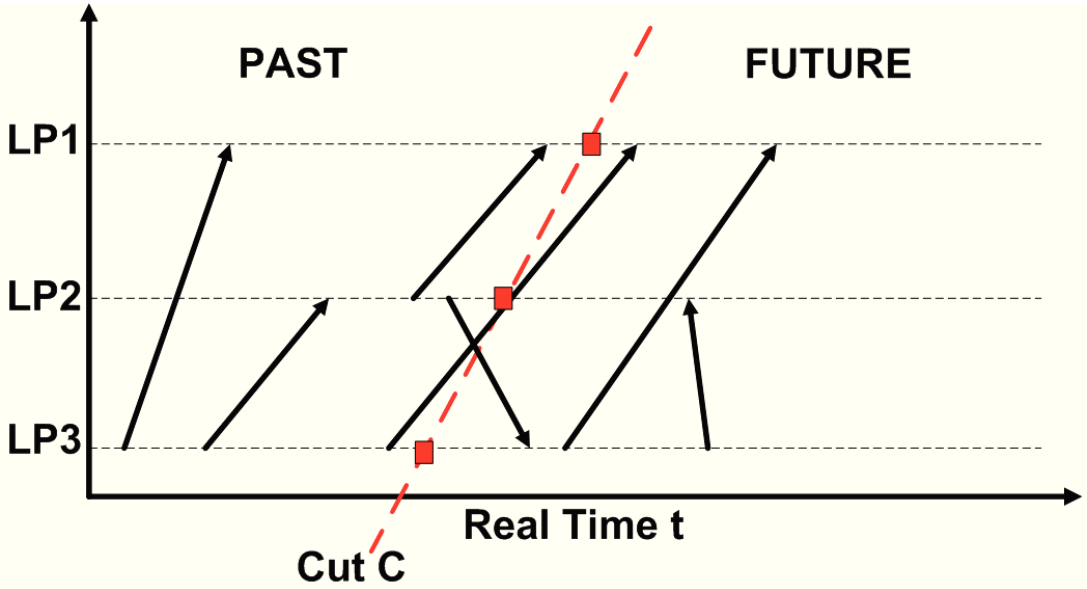
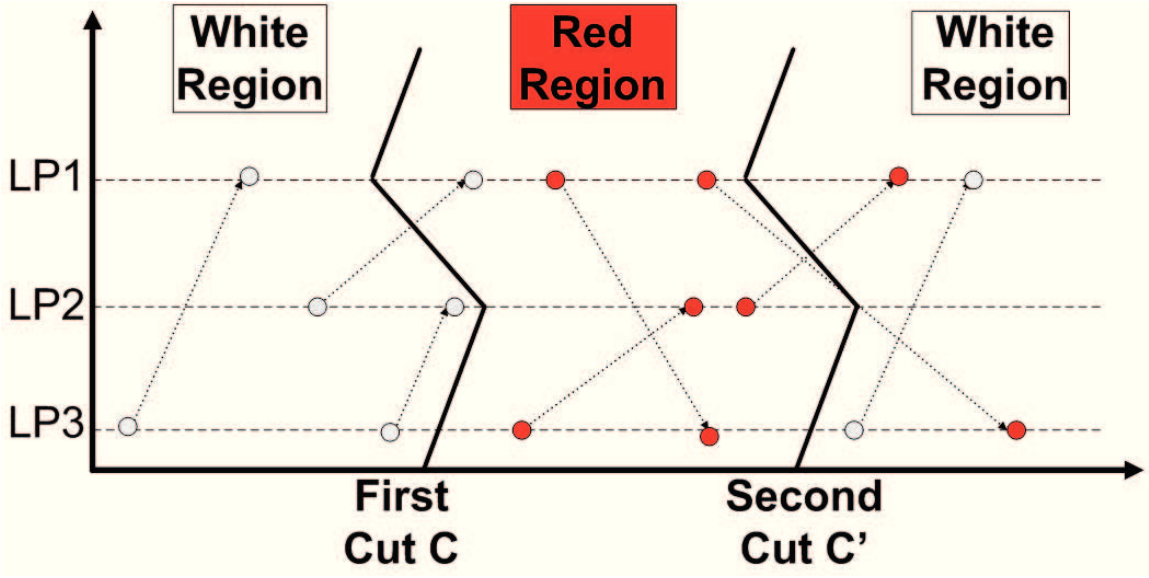


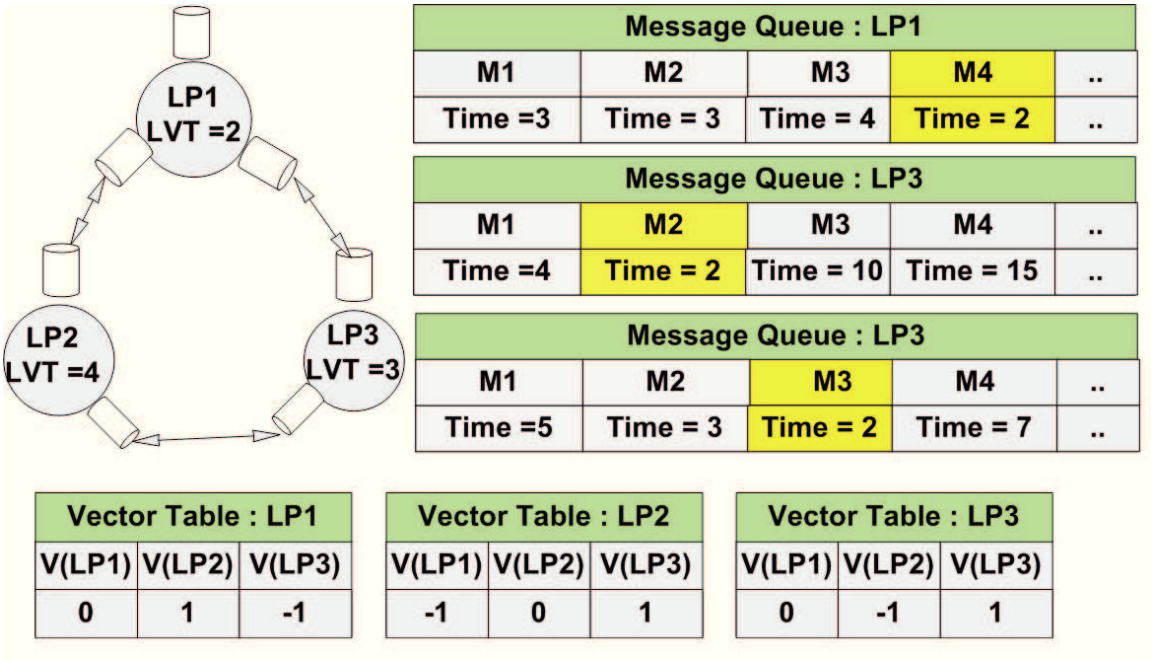
Figure 3.6: A cut in real time across LPs separating PAST and FUTURE events.

So, the algorithm defines two cuts C and C' (C' happens after C). All the messages are attached a colour(red/white) to indicate the state of the cut (First/second). By default, all the processes and messages are white, but when the cut C at time t is initiated, all the processes and messages sent after that will be red (see figure 3.7). Each process maintains vector tables that is a statistical record of number of white messages sent and received from other processes.

The concept of vector tables is used to identify the transient messages. Each Logical process maintains vector counts of messages sent and received from other processes. For example in the figure 3.7, LP1 has sent 1 message to LP2 and has received 1 message from LP3. Whenever a LP receives a message from a remote process, it decrements the counter for that process and when it sends a message to another process, it increments the counter for that process. If there are no transient messages, then the final summation



(a) Two stages of cuts.



(b) Message queues and vector tables of LPs.

Figure 3.7: GVT calculation.

of all the counters of all LPs will be zero. Otherwise, some LPs have to wait for transient messages.

An initiator is assigned to the system, who is responsible to initiate the algorithm, gather global state, set a horizon (GVT) and inform GVT to all processes. The initiation of the algorithm depends on the requirements of the system. In PDES-MAS, always root

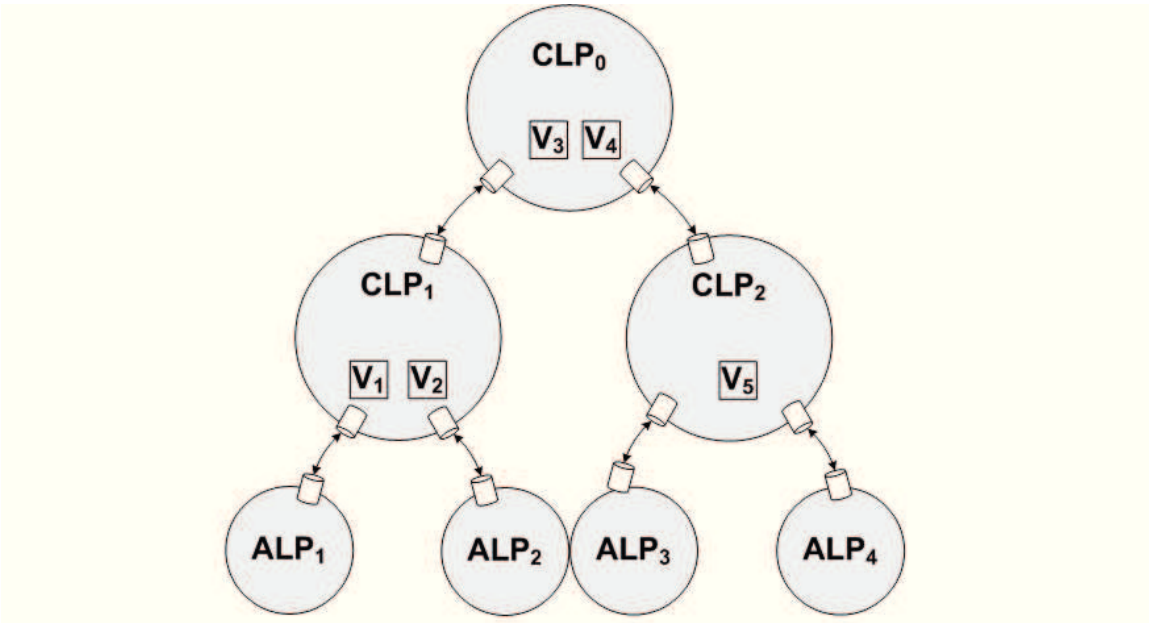
CLP tree is assigned as the initiator. Other CLPs can send a request message to initiate the algorithm. When the first cut is initiated, all the processes are set to red, attaches its local clock (*MIN(LVT, latest Red Messages)*) and their vector tables, then send back to the initiator. The initiator then calculates the minimum clock of all the processes and if the vector table is fine, then the second cut C' gives the horizon (GVT) to all the processes. If there is any anomaly in the vector table (like any process have to wait for white transient messages), then it has to initiate another round to fix it. Once all the anomalies are removed (like all white messages are received), the horizon (GVT)) is set to all processes, which is used to remove all the states with time $<$ GVT.

3.2.6 Routing

The system should route queries and updates from ALPs through the tree to the CLP that hosts and maintains SSVs As already mentioned, a message arriving at a port in a CLP will be either processed locally (port - H) or forwarded through any of the external ports (U, R, L). A CLP uses its routing tables to forward any message to its destination port. The routing tables are constructed during the initial stage of the simulation.

A *Routing Table* in a CLP, as the name suggests, is used to forward a message to reach the location of a SSV/CLP/ALP through any of the ports. The tables are constructed with the formats as *(CLP id, Port)*, *(SSV id, port)* and *(ALP id, port)*. Basically, the knowledge of binary tree format with number of CLPs and ALPs in the tree is known in advance. The initial locations of SSVs in the tree are identified with the XML file using *InitialStateParser* module. With this information, a CLP is aware of its immediate neighbours through which the routing table *CLP id, port* is constructed to reach every CLP in the tree. Each SSV is associated with a CLP and is used to construct *(SSV id, port)* table. The locations of ALPs (always attached as leaves) in the tree is identified using strict binary tree configurations. So, each ALP is identified with the location of its *parentCLP*. A parentCLP is the CLP to which an ALP is attached. So, requests from ALPs are forwarded through the CLP tree using these routing tables. The response

message follows the same route path to the ALP using its parentCLP id. The figure 3.8 presents 3 CLPs tree topology of the framework with SSVs in the tree at the top and corresponding routing tables of CLP id 0 (or root CLP) are at the bottom.Within an ALP, there are no such algorithms required and just forwards the requests to its parentCLP.



Routing Tables – CLP0

SSV	Port
1	L
2	R
3	H
4	H
5	R

CLP	Port
0	H
1	L
2	R

ALP	Port
1	L
2	L
3	R
4	R

Figure 3.8: A tree of 3 CLPs and 4 ALPs (top) with the routing table of CLP id 0 (bottom).

3.2.7 Message Passing Interface

In a distributed environment, LPs communicate with each other using messages. As the scale of the system increases, the bandwidth consumption will also increase. So, the communication has to be efficient, less time consuming and easier to understand. PDES-MAS is implemented with *Message Passing Interface (MPI-2)*, a standard *Application Programming Interface (API)* for communication between LPs. *LAM/MPI (7.1.3)*[131] and *openMPI (1.4.3)*[40] libraries are both intended to provide an implementation of MPI for high performance parallel computing architecture. Both provides mechanisms to launch and run processes in different hardware cluster platforms.

PDES-MAS framework is implemented using C/C++ programming language and is compatible with both *LAM/MPI (7.1.1)* and *openMPI-1.4.3* libraries. Several MPI functions are used in this framework to perform a specific task. *MPI_barrier* function in a LP is used to block until all other LPs reach the same point. It can be used as a check point for all LPs to coordinate and finish routines properly. In PDES-MAS, this routine is used during *Initialisation* to make sure all ALPs and CLPs have initialised properly before the simulation run. Each LP is identified with a unique *rank-id* within the *MPI_COMM_WORLD* (communication world for all MPI processes). Within this framework, the rank-id is used to assign a type of LP (CLP/ALP). For example, if the experiment setup is launched with 7 processes (3 CLPs and 4 ALPs), then process ids from 0 to 6 is assigned to each LP like (0 - 2) for 3 CLPs and (3 - 6) for the remaining 4 ALPs.

MPI-Interface module uses two MPI methods, *MPI_ISend* and *MPI_Receive*. *MPI_ISend* method is a non-blocking call which sends the message immediately from the buffer. It allocates a communication request object and stores the handler to check the status of the operation. *MPI_Test* method is used immediately to check its status, for example, the status of handler is true if the operation *MPI_ISend* is complete. *MPI_Receive* method receives the message from other LPs which is then added to the buffer for further processing. In a small experiment of 2 LPs sending messages using *MPI_ISend* and *MPI_Receive*

methods, the time taken for a message to travel from LP1 to LP2 and get back to LP1 is on average is 6 μsecs. This depends on the size of the message protocol but protocols defined in this framework do not pose any concern.

3.3 Summary

This chapter presented an overview of PDES-MAS framework and how simulation of MAS is modeled within the framework. Agents are assumed to be situated, embodied and mobile moving in a dynamic and non-deterministic environment. PDES-MAS is built on PDES paradigm to distribute and run parallel simulation of MAS. It models MAS environment objects as a list of SSVs. Each SSV maintains a list of write periods evolved over time > GVT. PDES-MAS employs optimistic synchronisation to make sure that the events adhere to the Local Causality Constraint (LCC). It employs rollback mechanism to correct the ordering of events. ALPs model agents actions and CLPs maintain the shared state of the simulation where in ALPs access them using queries and updates in a time synchronised manner. Modules within PDES-MAS are initialised using XML files and agent trace files. Preliminary experiments show that the initialisation procedure (parsing all information to construct tables) is not an overhead as it takes 0.3 msec to initialise a SSV in a CLP, 0.05 msec to load a trace in an ALP and the total initialisation time takes just 235 msec. Also MPI libraries take just 6μsecs to send and receive back a message. The system presented so far can access data using ID query and write with the assumption of a universal knowledge of all SSV IDs in the simulation. However, the following chapter present a notion of synchronised range queries to access a set of SSVs in a time synchronised manner.

CHAPTER 4

RANGE QUERIES IN PDES-MAS

This thesis presents a notion of a synchornised range query within the implementation of PDES-MAS framework. The assumption of PDES-MAS system is that the IDs of SSVs are known universally across the agents in the simulation, where they can read and write in a time synchronised manner. This thesis presents uses the notion of a range query to access a set of variables that match a range predicate. For this purpose, the algorithms are designed and built on the working implementation of PDES-MAS kernel. This chapter presents a detailed overview of the design and analysis presented in [138].

4.1 Design Outline

The PDES-MAS framework is based on PDES paradigm, where a simulation model is divided into a network of concurrently executing Logical Processes (LPs), each maintaining and processing a disjoint state spaces of the model. Two types of LP exist in a PDES-MAS simulation. *Agent Logical Processes* (*ALPs*) are responsible for modeling the behaviour of the agents in the MAS. This includes the processing of sense data, the modeling of behavioural processes (such as planning or rule set 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 3.2.

CLPs essentially implement a space-time Distributed Shared Memory (DSM) model whereby agents interact with their environment and communicate via accessing public variables. Each CLP has four ports namely (Here, Up, Right, Left) connected with neighbouring LPs. A standard 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. The design options for implementing this operation in a more efficient manner were explored in [31]. This section details the implementation of one such design - a Range-Based approach - in a logical-time synchronised manner.

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 shown in Figure 4.1 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 the responsibility

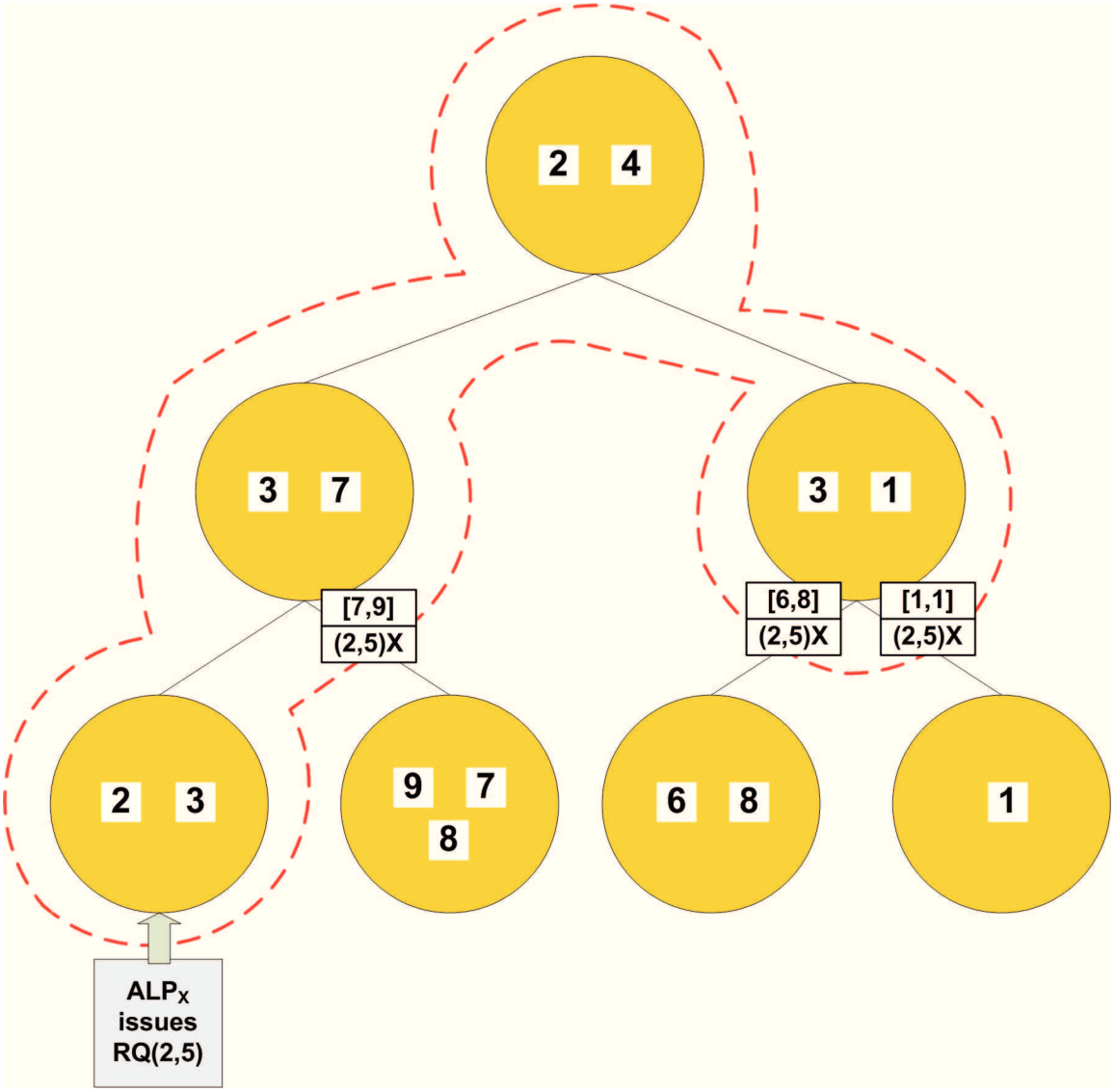


Figure 4.1: The propagation-horizon created by a Range-Query traversing a tree of CLPs

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.

The figure 4.2 presents an evaluation of scenarios (an SSV change inside and outside the horizon causing a port-range change on the horizon) causing both rollback and non-rollback conditions.

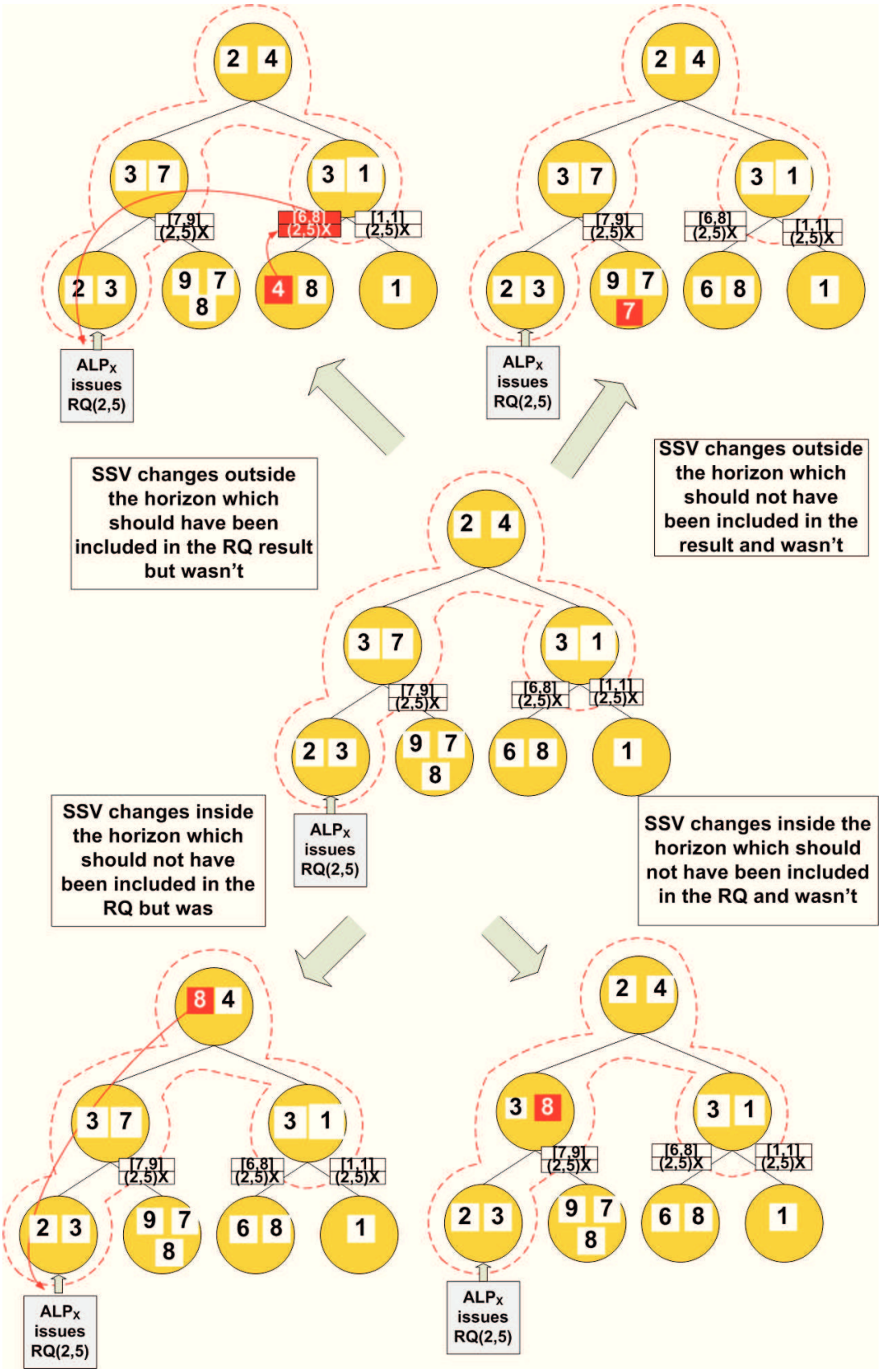


Figure 4.2: Reactions to SSV changes inside and outside of a propagation-horizon. Changes inside are directly detected by the SSVs the query accessed. Changes outside are detected by range-updates to ports lying on the horizon.

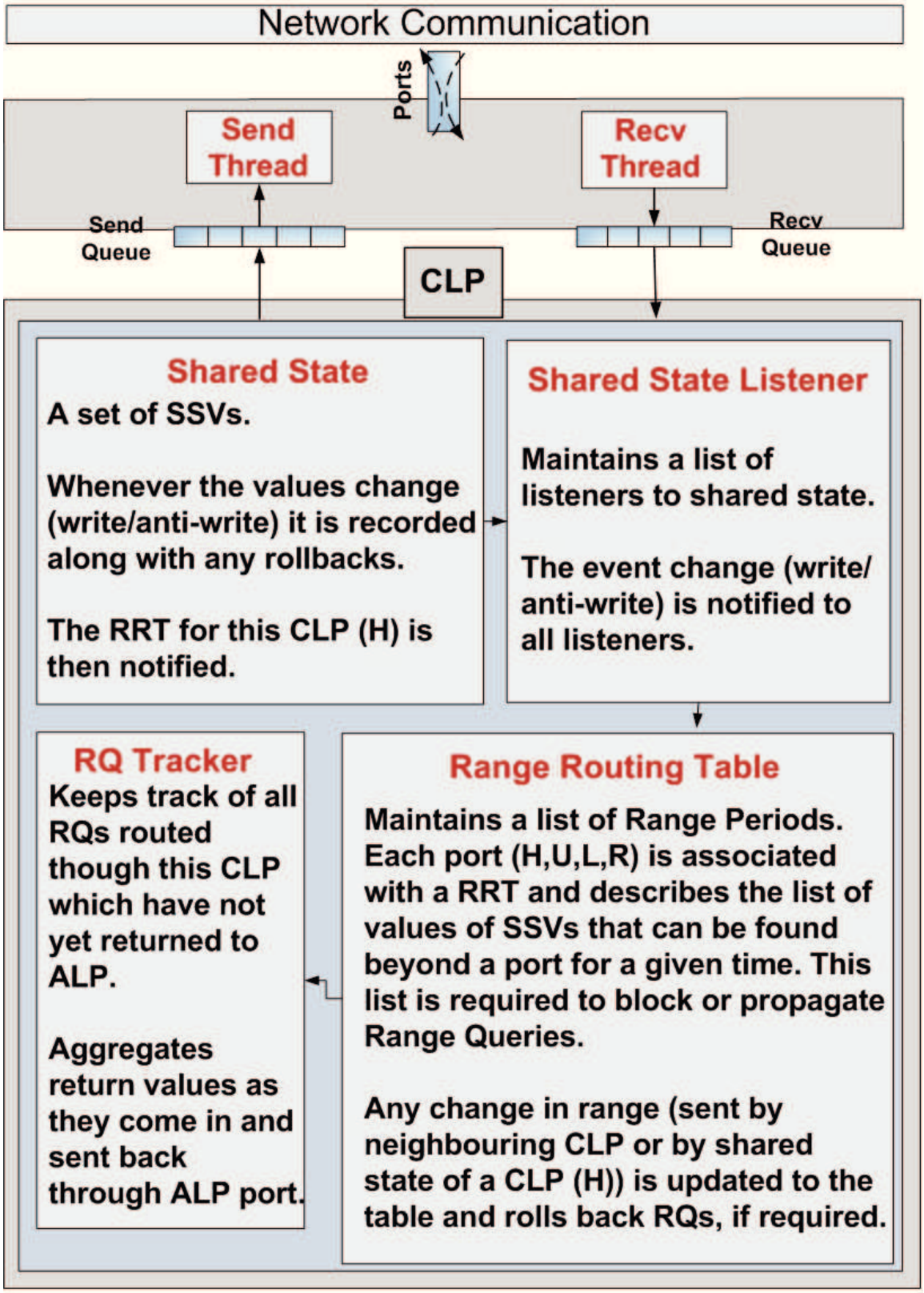


Figure 4.3: An overview of modules handling range queries within a CLP.

4.1.1 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 it requires a more complex algorithm than the simple (min,max) comparison. Before defining this algorithm some data structures and terminology have to be defined:

- 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 *RangePeriodList* 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 RangePeriodList 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 RangePeriodList covering the set of SSVs in *all* CLPs beyond this port. These three port RangePeriodLists 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 Range Queries arrive, the RangePeriodLists 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.

2. As write and anti-write messages are received by the CLP, the RangePeriodList H must be altered, if necessary. When H changes, the port RangePeriodLists U, R and L must also be checked and, if necessary, these updates are propagated to further CLPs.

3. As RangePeriodLists 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.

An overview of modules in a CLP handling these concurrent processes is depicted in figure 4.3. There are three modules in this framework, namely, *Range Routing Table (RRT), RQTracker* and *SharedStateListener* to handle range queries, updates and roll-backs in a time synchronised manner. A detailed working of these modules is presented in the following sections.

4.1.2 Maintaining Range Period List

Range Routing Table (RRT) maintains a list of range periods at each port (H, U, L, R) and send updates to neighbouring CLPs (triggered indirectly by neighbouring CLPs or by shared state of a CLP (H)). This section presents an algorithm for maintaining range period list. 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 a 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 4.4. A snapshot of Range Period List corresponding to a tree structure is depicted in figure 4.5.

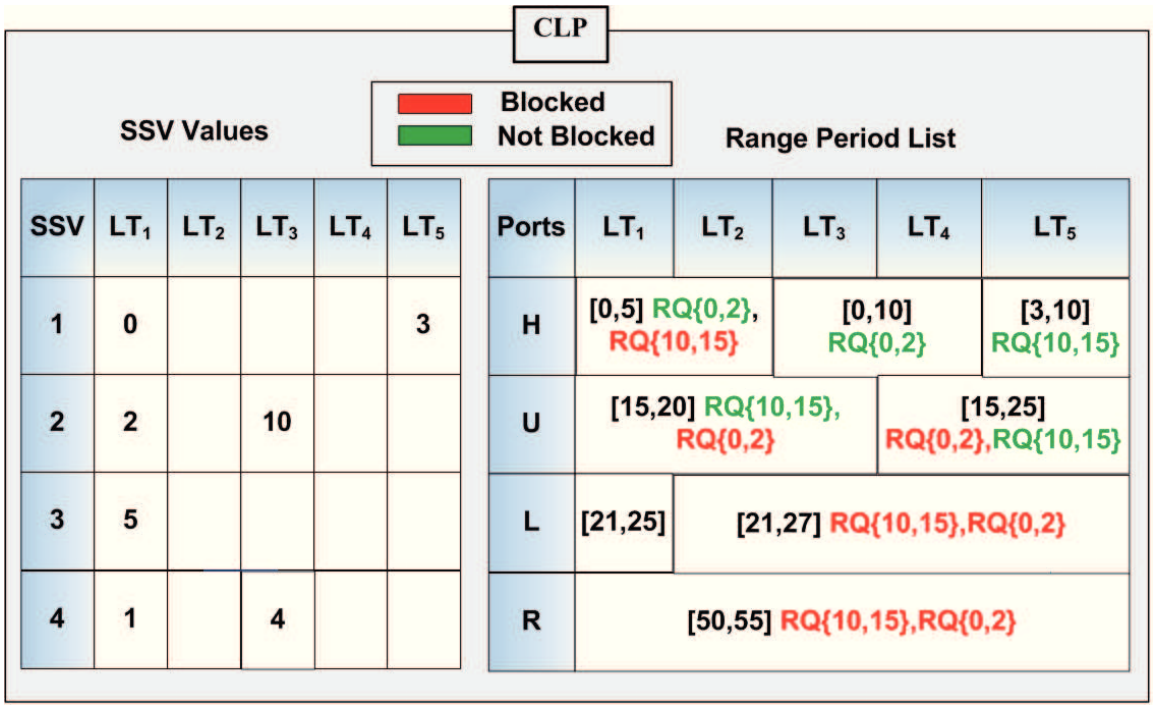


Figure 4.4: An example of a Range Period List (at the right) of a CLP at ports H. L, U, R evolved over time.

Any change in the value of a SSV is indicated to RRT maintaining Range Period List at port H using Shared State Listener module. Shared State Listener module is an event listener to shared state of a CLP at port H. All events such as read, write, anti-write, anti-read are indicated automatically to objects registered with this module such as Range Period List at port H, Access Monitor (monitors the access patterns on SSVs, see chapter 5 for more details). In the case of maintaining Range Period List, only write/anti-write events are indicated. The Range Period List data structure is maintained correctly using the algorithm depicted in figure 4.6.

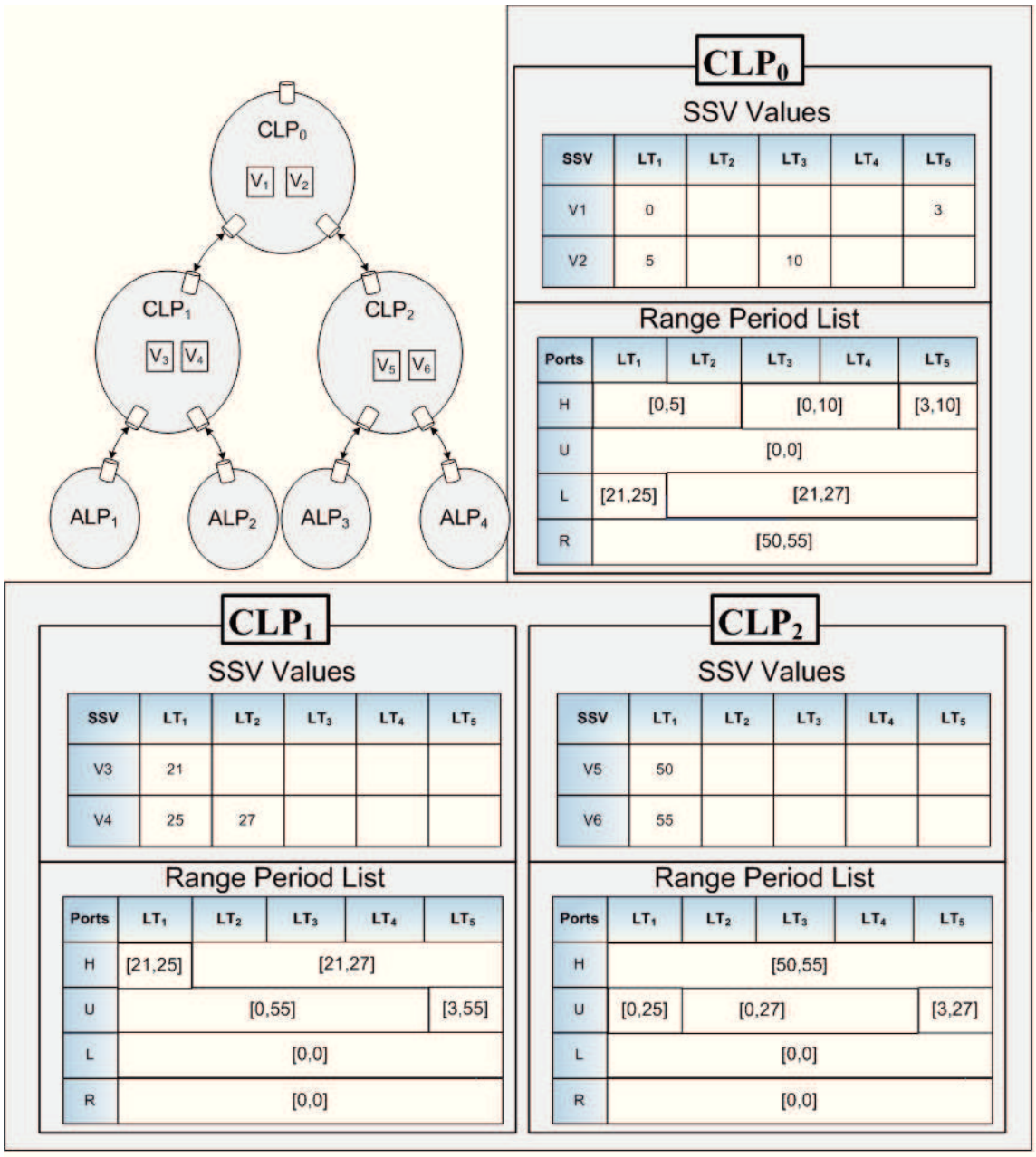


Figure 4.5: A snapshot of a Range Period List of CLPs (0, 1, 2) at ports H. L, U, R evolved over time.

This algorithm can be optimised using heuristics but for clarity is presented in its brute-force form here. It has the effect of recalculating the entire RangePeriodList beyond the write/anti-write logical time. This process is seen for a sequence of updates to an example SSV set in figure 4.7.

```
def update_range_list(time):
    # find range period split by the write/anti-write
    rp_split = find_rp(time)
    if (rp_split != NULL):
      # calculate the range now covered by the
      # SSVs at this time
      new_range = get_range(ssv_set, time)
      if (new_range != old_range):
        # insert a new RangePeriod for this time
        insert_rp(rp_split + 1, time, new_range)
        # for all RangePeriods > time, recalculate
        # their ranges
        for (rp in range_periods(time)):
          if (rp.range != get_range(ssv_set, rp.time)):
            update_rp(rp, get_range(ssv_set, rp.time))
    else:
        insert_rp(start_of_list, time, new_range)
```

Figure 4.6: The algorithm for correct maintenance of a RangePeriodList data structure over a set of SSVs

4.1.3 Range Update Propagation

This section presents an algorithm to propagate synchronised updates to neighbouring CLPs. Whenever the algorithm depicted in figure 4.6 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. The update propagation freezes at a port beyond which no other ports$'$ range information are affected. This relationship between list modifications and checks is depicted in the matrix in figure 4.8.

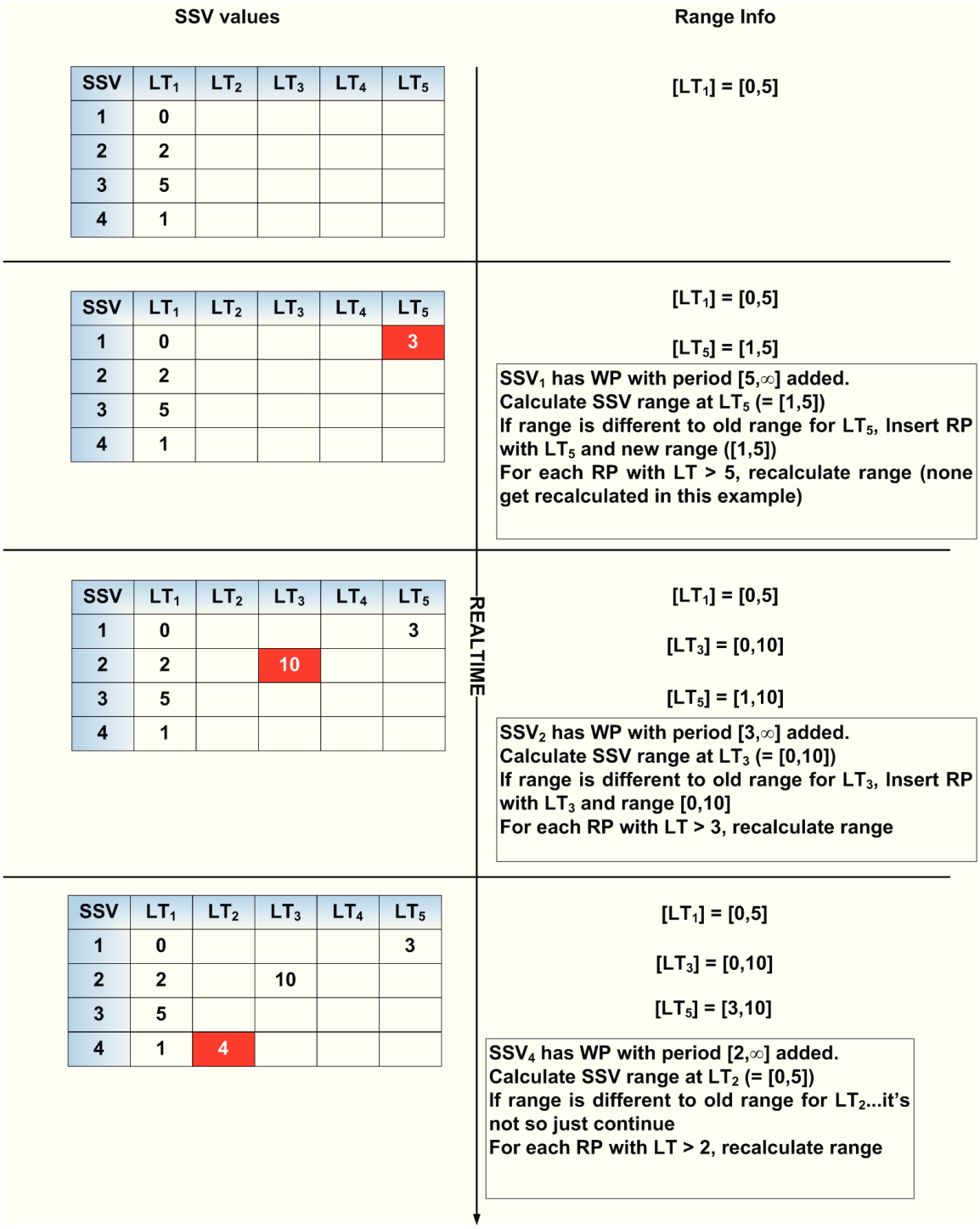


Figure 4.7: A sequence of updates occurring to a set of SSVs in real time (top to bottom) and the updates made to the RangePeriodList in response. For each update the algorithm execution is detailed.

```
# the method is invoked with a list of new range
# updates and the origin port
def send_range_update(range_update_list, origin_port):
    # iterate through the list of range updates.
    # Each range update has
    # a range associated with a time period.
    for(ru in range_update_list):
    # iterate through CLP ports except origin port
      for(port in CLP.rem_ports):
        old_range = find_range(port, ru.time)
        # recalculate the new range using the
        # logic mentioned above.
        new_range = calculate_range(origin_port,
                ru.range, CLP.ports)
          if(old_range != new_range):
                    send_range_update(port, time,
                    new_range)
```

Figure 4.9: The algorithm to propagate synchronised range updates

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)

Figure 4.8: The relationship between RangePeriodList modification and recalculation.

The algorithm to handle synchronised range updates arriving at any port (H, U, L, R) is depicted in figure 4.9.

4.1.4 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 4.4. 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 4.1.

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 4.10. Once results from all ports are aggregated it is sent back to the originated port and the entry is deleted.

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

Figure 4.10: A snapshot of RQTracker table.

4.1.5 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 4.2) 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 4.1.4, 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.

An example scenario based on the range period list depicted in figure 4.4, the range query (10,15) arriving at logical time 5 is processed at port H and U and blocked at ports L and R within a CLP. Suppose, a straggler write (arriving late in real time) to a remote variable has changed its value such that the range query blocked at say port R previously

```
def update_ssv(time, value):
    # find the write period split by this write
    old_wp = find_wp(time)
    if (old_wp != NULL):
        new_wp = insert_wp(old_wp, time, value)
        # for all RQs > time, evaluate rollback conditions
        for (RQ in old_wp.queries):
          if (RQ.time > time):
            old_wp.remove(RQ)
            # either rollback query or place in new
            # write period
            if (!RQ.contains(old_wp.value) &
                !RQ.contains(value)):
              new_wp.add(RQ)
            else:
              rollback(RQ)
    else:
        insert_wp(start_of_list, time, value)
```

Figure 4.11: 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.

should be processed again and allowed to progress through port R. In this case, the range query (10,15) will be rolled back and re-issued to get back with the correct 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 algorithm is shown in Figure 4.11 and depicted for the case of Range Updates in figure 4.12.

4.2 Evaluation

This section presents an evaluation of synchronised range queries algorithms presented in section 4.1. The algorithms are implemented within the PDES-MAS framework using C++ programming language. The framework is fed with the traces of agents' actions in

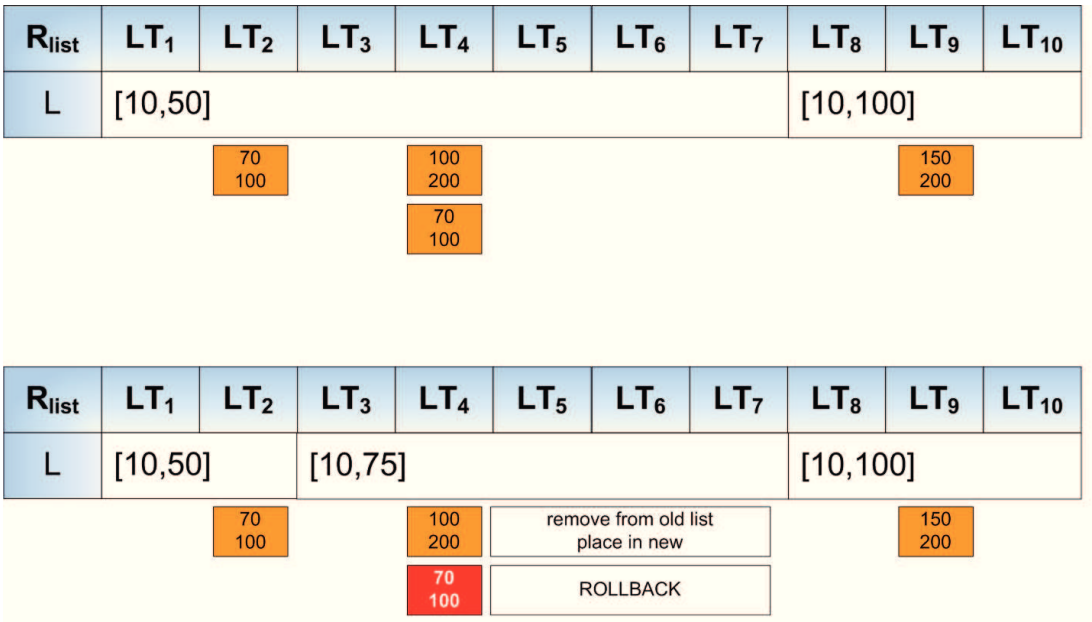


Figure 4.12: How updates of Range Lists lead to rollbacks of Range Queries (using the algorithm in Figure 4.11). Above two Range Lists are shown with records of four blocked queries at times 2, 4 and 9. Below, a Range Period update at time 3 leaves one query having been routed correctly, and one incorrectly. The latter is rolled back whilst the former is simply moved from the old Range Period to the new one.

the simulation. The traces themselves were generated using various models implemented with the MWGrid multi-agent simulation toolkit under development at the University of Birmingham (see `http://www.cs.bham.ac.uk/research/projects/mwgrid`). The agent actions such as ID read, ID write and range query are translated into event traces format pre-defined in this framework. The trace formats for ID read and ID write are defined in chapter 3 and in section 3.2.1. The trace format for range query is given below

$$< agent\_id, rangequery, minimum\_range, maximum\_range, timestamp > \qquad (4.1)$$

The trace format is quite self-explanatory which has an agent id, action is range query with minimum and maximum range window and a timestamp. The experiments presented in this section are designed to analyse the performance over various metrics of

the concurrent processes of: Range Query propagation; Synchronisation via Range Query Rollback; and Range Update propagation in response to write/anti-write patterns.

4.2.1 Range Query Propagation

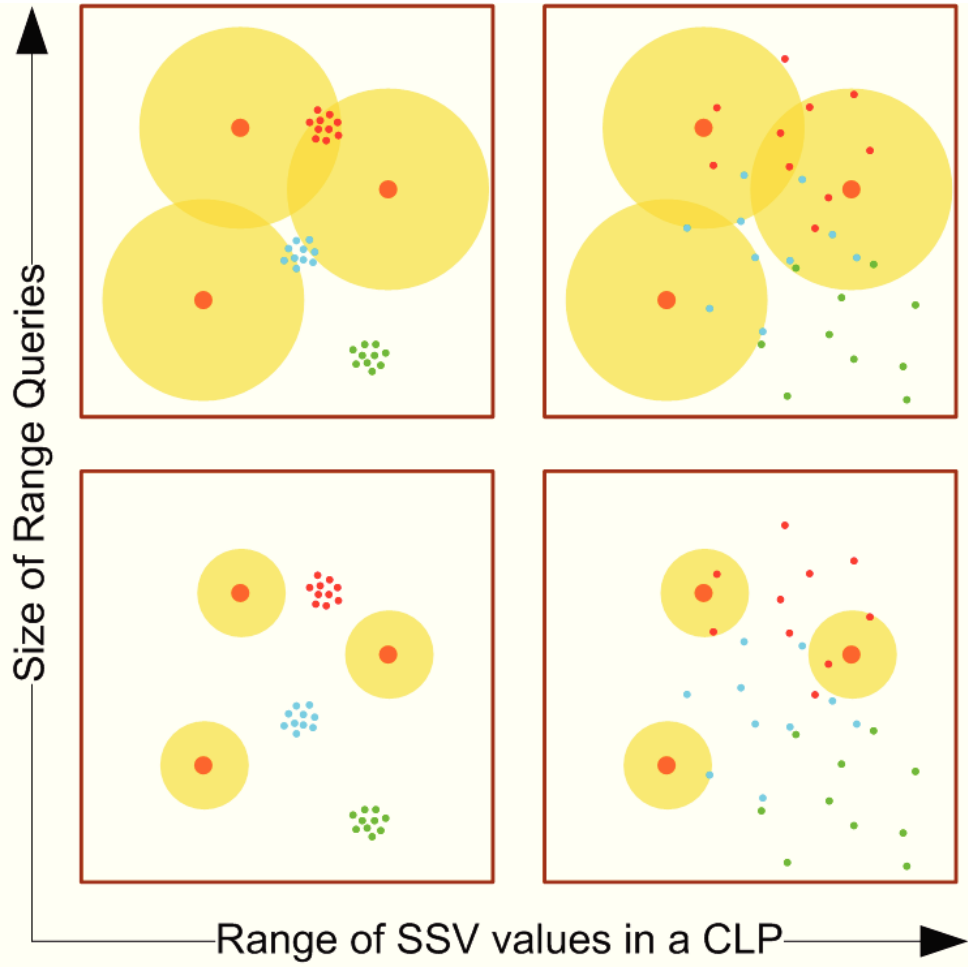


Figure 4.13: The logical view of the traces used to investigate the extent of Range Query propagation under different conditions. The SSVs maintained by each CLP share the same colour.

The first experiment analyses the extent of propagation of a Range Query (how far a query travels through the graph) as a function of two variables: the size of the query; and the range of values covered by each CLP, both measured as a proportion of the size of the simulated world. For this experiment a trace was generated in which:

- 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 CLPs (two attached to each leaf CLP) each issue a 2-Dimensional Range Query once per tick with a static position and radius. The static positions of agents are uniformly distributed on each experimental run. The radius corresponds to the *RQ size* parameter.

The logical view of the simulation layout corresponding to different values of these two variables is shown in figure 4.13.

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 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 4.14 and 4.15 illustrate, the average number of hops and wallclock time per query confirm the relationship of both Range-Query size and SSV value range to the propagation extent of Range Queries. The graphs do not demonstrate a complete linearity for all values of *RQ size* and *SSV range* (e.g.., with *SSV range* = 0.5, hops and wall clock time decrease from *RQ size* = 0.1 to 0.2). This is due to the initial distribution of SSVs in the tree, in relation to the agentś position in the virtual space. With the introduction of uniform randomness to agents positions, trend deviations are observable with smaller changes in the values of experimental parameters (as, for instance, in the case of SSV range = 0.5, where hops and wall clock time decrease from RQ size = 0.1 to 0.2) whereas a linear trend emerges with increasing values. However, the result bears out our expectation that the number of hops and wall clock time follow the same trend.

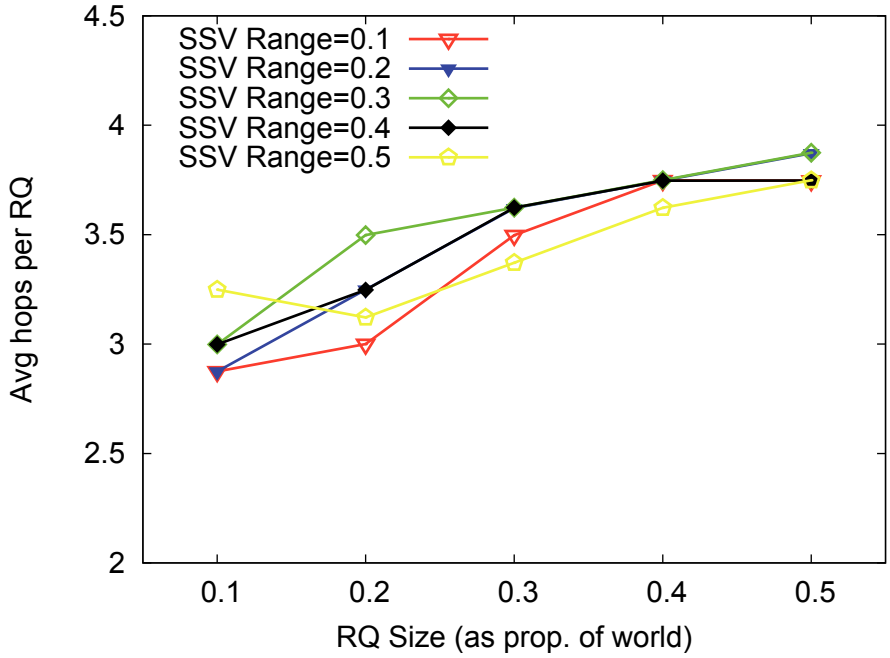


Figure 4.14: Average Hops per Range Query for Different Sizes of Query Performed by an ALP and different Ranges of SSV values held by a CLP

On average propagation extent increases 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).

4.2.2 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:

- *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

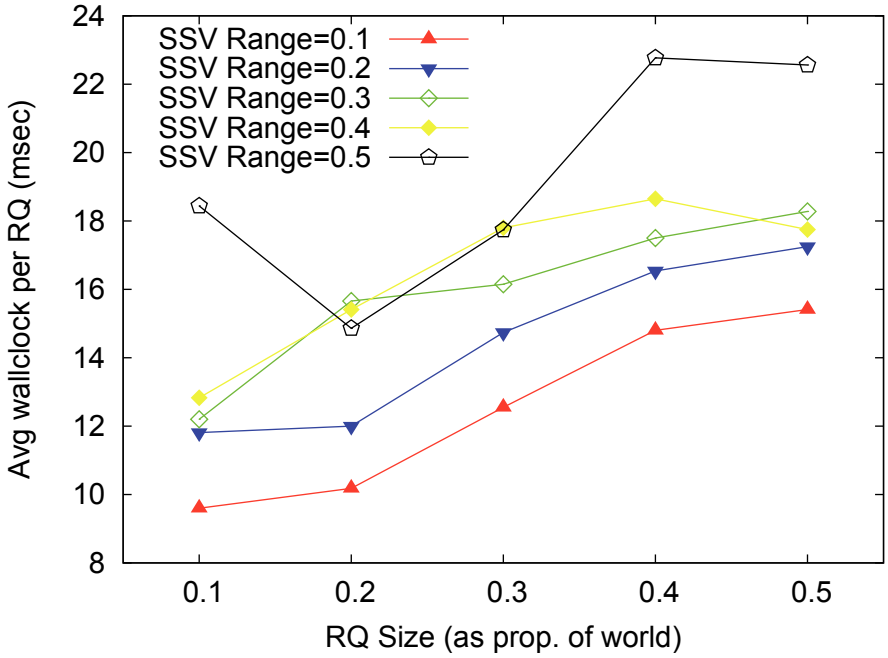


Figure 4.15: Average Walltime per Range Query for Different Sizes of Query Performed by an ALP and different Ranges of SSV values held by a CLP

write delta is the metric distance between the SSVs current value and the new value that is written.

Range Updates triggered 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 a 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'.

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

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[1]. 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.

4.2.3 Rollback Rate

The final experiment 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 4.2.1 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. The rate varies from 8130 (with *RQ size* = 0.1) to 8483 (with *RQ size* = 0.5). In the absence of other variables we expect this property to be proportional to wallclock time as it ranges from 49 (with *RQ size* = 0.1) seconds to 98 (with *RQ size* = 0.5).

[1]Note that the local set changing will not necessarily cause the propagation of a Range Update message to a neighbouring CLP, see section 4.1.1.

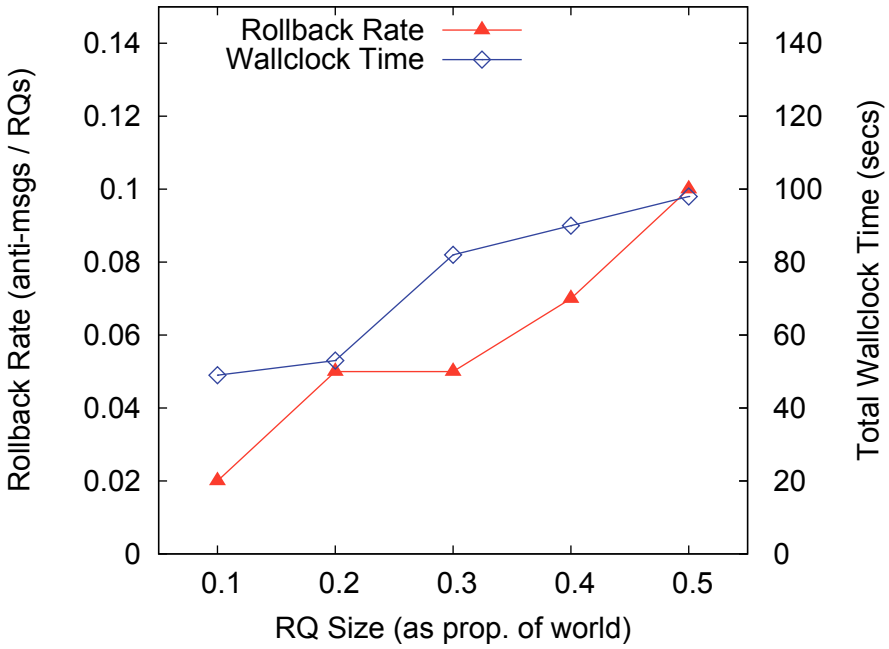


Figure 4.16: The Relationship Between Range Query Size and the Incidence of Rollback, and its Effect on Execution Time.

The correlation between Query Size and Rollback Rate and the corresponding impact on total execution time is shown in figure 4.16 and strongly agrees with the predictions given above.

In general these experiments indicate the central truth that this 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 a risk of rollback due to straggler updates in the data structure.

4.3 Summary

This chapter has presented a detailed design for logical-time synchronised Range Queries in a parallel simulation kernel. 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 what values lie beyond a given edge in the graph. Evaluation from this design shows the perspective of the volatility of the data structure under different query and update loads and its performance in terms of service latency and execution time under the same loads. There is a strong correlation between the data structure's volatility and its performance as a simulation kernel.This points out the requirement to adaptively reduce the service latency and execution time.

CHAPTER 5

RANGE QUERIES IN THE PRESENCE OF STATE MIGRATION

The current implementation of PDES-MAS configures CLPs in a static tree-structure and SSVs are uniformly distributed across the tree. Several approaches of achieving distribution within PDES-MAS system are reported in [79]. A competitive optimisation algorithm to SSV migration for PDES-MAS framework is reported in [105]. SSV migration is achieved with migrating SSVs closer towards the accessing agents. However, it has been built on the assumption of ID query based PDES-MAS system. Here, we adopt the mechanism to operate in the presence of Range Query based PDES-MAS system. The aim of the strategy is to reduce the service latency (query response time) and execution time of the PDES-MAS system. The design and experimental results presented in this chapter are reported in [137].

5.1 Migration Strategy in PDES-MAS

In this section, we present a survey of approaches to data distribution management within PDES-MAS system. PDES-MAS maintains the shared state of the simulation system in a CLP tree and ALPs get the responses using the services (such as reads or writes) available to them. One way to balance is to re-configure the CLP-tree. The number of CLPs can be increased or decreased dynamically based on the load incurred from the ALPs. The

figure 5.1 depicts an example of such re-configuration of CLP tree. One CLP configuration is split into three and then merged back to just two. Also the ALPs are assigned to a CLP according to their access patterns.

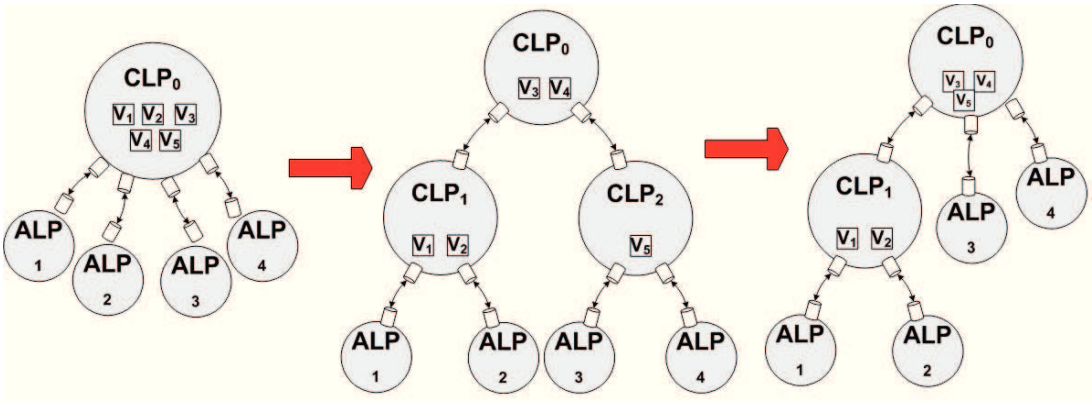


Figure 5.1: Dynamic re-configuration of CLP tree.

Second method is to migrate ALPs and/or SSVs through the CLP tree, keeping number of CLPs fixed throughout the simulation or changing both configurations. The figure 5.2 depicts an example of migrating ALPs through the CLP tree. However, this framework has adapted SSV migration, where the assumption is that the CLPs in the tree and ALPs are in fixed locations and migrate a set of SSVs closer to the ALPs that access them the most. For this purpose, several optimisation algorithms are reported at [105] for the PDES-MAS framework.

The state migration approach presented in [105] uses ports of CLPs to monitor the access patterns of agents. It populates access information (queries and writes) of agents at each port on each SSV within a CLP. The idea is to reduce the access cost and simulation time of the system. A total access cost of a SSV is a function of the number of accesses and hops for each access. 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 queries and updates (the number of accesses) and the number of hops needed to reach them in the CLP-tree. SSVs are migrated towards the port that has the highest cost of all ports in a CLP. The algorithm also provides mechanisms to swap loads between CLPs to avoid bottlenecks. A Migration of a SSV is achieved by deleting the SSV from

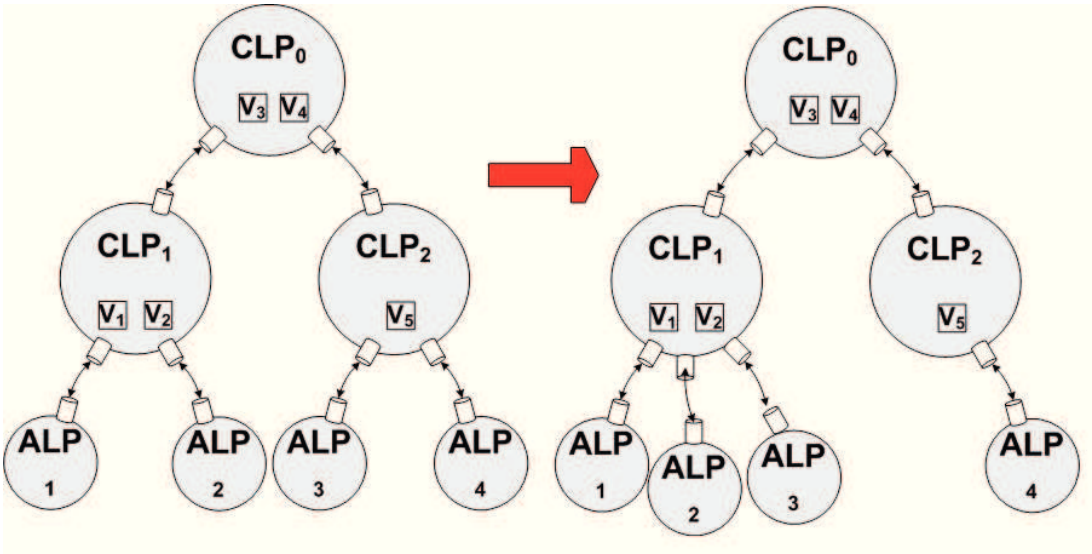


Figure 5.2: Migration of ALPs through CLP tree.

one CLP and adding to the neighbouring CLP together with its entire history, namely the list of *Write Periods* (A write period is the validity period of a value with a start and end time). The ports of both CLPs can be easily updated with the SSV's new location. This does not affect processing any transient simulation messages such as ID Query/write operations.

However, this basic mechanism is adapted to migrate SSVs in the presence of synchronised Range Queries algorithms. This approach is called *State Migration* which aims to reduce the propagation of synchronised range queries thereby reducing access latency and execution time of the simulation. We use threshold parameters to reduce the frequency of state migration. The migration presented in [105] is complicated by the fact that each CLP additionally maintains *Range Periods* of a set of SSVs evolved over time (A Range Period is the validity period of range information of a set of SSVs, see section 4.1.1).

Migrating an SSV with its write period history also means changing the range period history at both CLPs. To achieve this, each range period evolved over time $>$ GVT has to be evaluated against the entire list of write periods of SSVs selected for migration. 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) (see section on update mechanism in section 4.1.2). Following the example in figure 4.8 depicting 4 SSVs and its write periods,

if SSV 3 is selected for migration, all range periods has to be evaluated and altered (if required). Each range period also has a list of range queries processed over the time period. Since the existence of range query is recorded at each port, it has to be evaluated against the list of write periods of SSVs selected for migration. The same procedure has to be repeated at the destination CLP where SSVs are migrated. The CLPs involved in this migration process has to be kept idle (blocking processing any simulation messages such as ID read/write/range query) until the range periods updated properly. But this procedure is too complicated and complex for the simulation framework. Alternately, a simple approach has been used to allow migration of SSVs without affecting any transient simulation messages. The algorithms are outlined in the following section.

5.1.1 Migration of SSVs

An overview of modules in a CLP handling state migration is depicted in figure 5.3. The decisions for migrations are all made without any regard to a particular time scope and all metrics involved in them are simply stored and iteratively updated as SSVs are added, read, written and removed. Essentially, the migration is achieved by monitoring the access patterns of each SSV at each port. If any SSV/SSVs exceeds a pre-defined access cost threshold, then the SSV is migrated towards the port that accesses the most.

After this, the migration module will calculate the set of SSVs to migrate and the ports down which they should be pushed. This is a two step process:

- Determine a set of SSVs which have a high enough total access cost to be considered and
- For each of these SSVs determine if it will be beneficial to reduce the access cost of a CLP.

An example of the data structure maintained by the migration algorithm is defined in table 5.1, which has 4 SSVs with their access costs. The three $Cost_X$ parameters measure

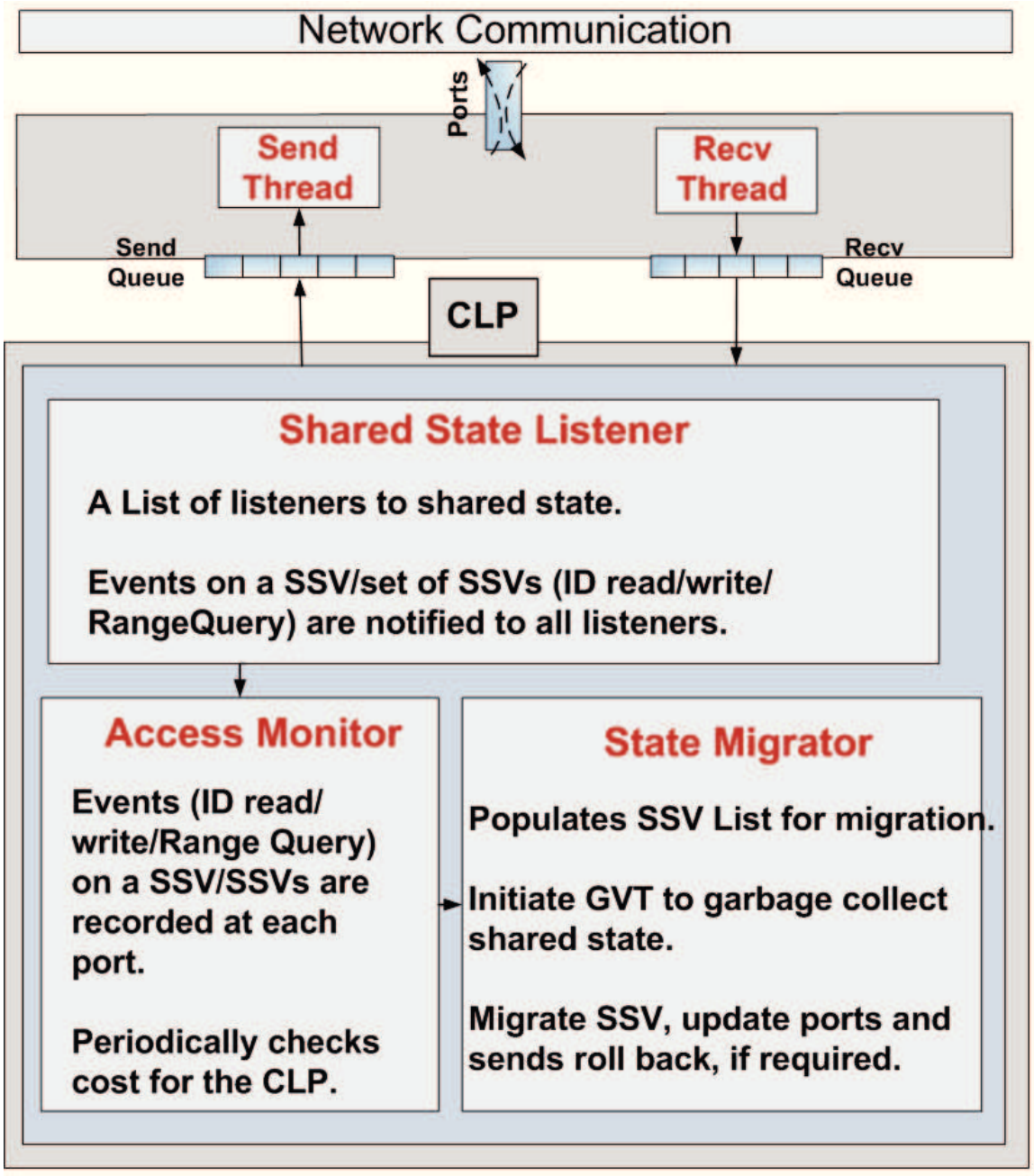


Figure 5.3: An overview of modules handling State Migrations within a CLP.

the cost of accesses through the port $_X$. Assuming in this example that the thresholds being used are: $TH_{load} = 45$ (A CLP must have a 'total' cost > 35 before the migration algorithm is triggered), $TH_{cost} = 10$ (An SSV must have a cost > 10 before it is considered for migration) and Port Choice Metric = Port$_X$ (chosen if $Cost_X >$ sum of all other port costs). The initial data structure for recording access costs of each SSV at each port is defined in table 5.1.

SSV_id	Cost_U	Cost_L	Cost_R	Cost_Total
1	5	5	5	15
2	5	10	0	15
3	0	8	0	8
4	2	2	2	6

Table 5.1: Initial access cost monitor data structure.

Suppose a ID read to $SSV4$ occurs from the port (R) and reports its hop count to this CLP as being 2. The table with updated cost to $SSV4$ is shown in table 5.2. This pushes CLP cost to 46, triggering a round of the migration algorithm. The algorithm rejects $SSV3$ and 4 (due to their lower access costs) and $SSV1$ (due to the fact no single port is identified to improve localisation). But $SSV2$ is selected for migration as its access cost at port (L) is higher than the sum of costs at other ports.

SSV_id	Cost_U	Cost_L	Cost_R	Cost_Total
1	5	5	5	15
2	5	10	0	15
3	0	8	0	8
4	2	4	2	8

Table 5.2: Updated access cost monitor data structure.

If a SSV is selected for migration, the state migration module handles the migration algorithm and 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. Mattern's GVT algorithm[86] is used in PDES-MAS (explained in detail in

chapter 3) 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 algorithm is depicted in figure 5.4 and an example of the migration algorithm is depicted in figures 5.5, 5.6 and 5.7.

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. The following section provide a quantitative analysis of the proposed approach.

5.2 Experimental Investigation

The overall objective of state migration is to improve the performance of the system by moving SSVs closer to the accessing ALPs. Moving SSVs closer to the accessing ALPs will reduce 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; each state

```
def 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)
def 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 write periods of migrating ssv
        DeleteWritePeriods (ssv)
        # update range periods accordingly
        UpdateRangePeriods (ssv)
        # send range updates accordingly
        SendRangeUpdates (time)
```

Figure 5.4: SSV Migration algorithm.

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 outweighs its inherent cost. Two experiments will be presented:

1. Investigating the impact of range query propagation and State Migration on the access cost of the system; and

2. Investigating the impact of increased rollback volatility and State Migration cost on average simulation time.

Experimental results both with and without state migration will be presented to allow for a clear comparison between the two configurations. An effort has been made to keep the experimental setup similar to the one used in section 4.2.1 so that the results of both papers remain comparable as well.

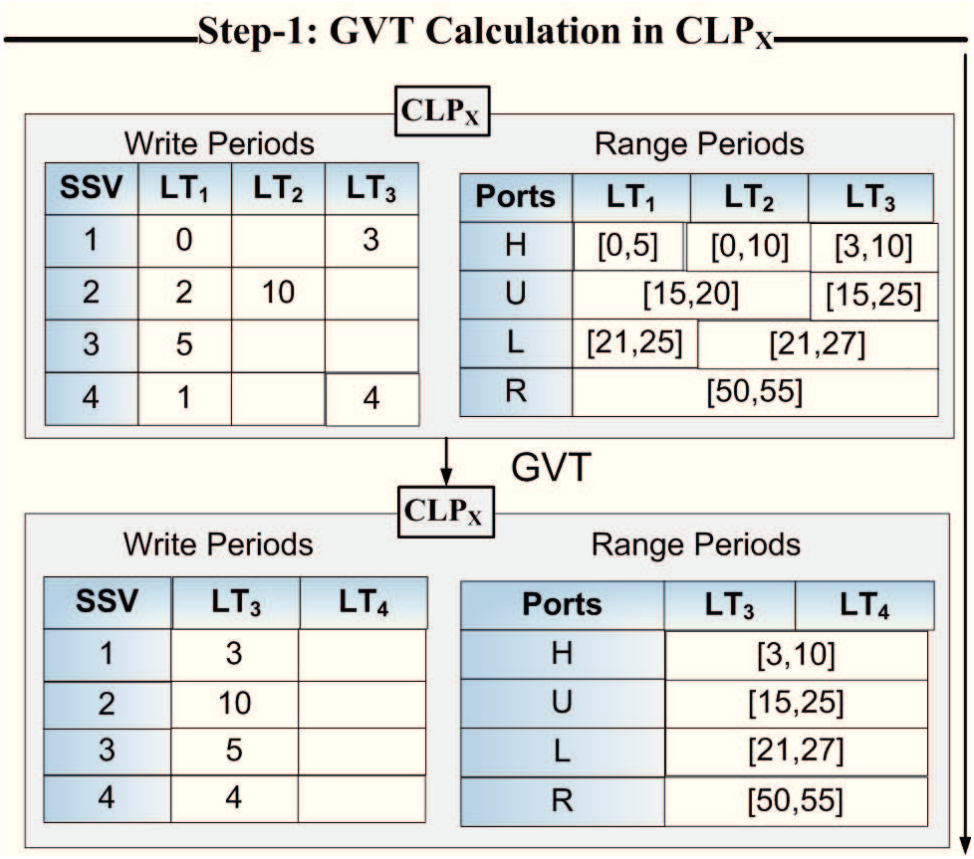


Figure 5.5: State changes in Write Periods and Range Periods table after GVT calculation.

The experimental platform included an implementation of PDES-MAS in C++ being fed simulation traces that provide read, write, and range query operations issued by ALPs. These traces were generated using variable models implemented with the MWGrid MAS toolkit under development at the University of Birmingham[1].

5.2.1 Experimental Setup and Parameters

The experimental setup used here similar to that presented in section 4.2.1 It consists of a CLP-tree of 7 CLPs, the root CLP of which has an initial 7 clusters of 50 SSVs. All SSVs were placed at the root CLP to more clearly show the effect State Migration has on performance. SSV values ($\langle x, y \rangle$) lie within a 2-dimensional radius around a randomly selected central point. The radius is determined by the *SSV range* experimental parameter. 8 ALPs, two attached to each leaf CLP are used to issue 2-dimensional range

[1] `http://www.cs.bham.ac.uk/research/projects/mwgrid`

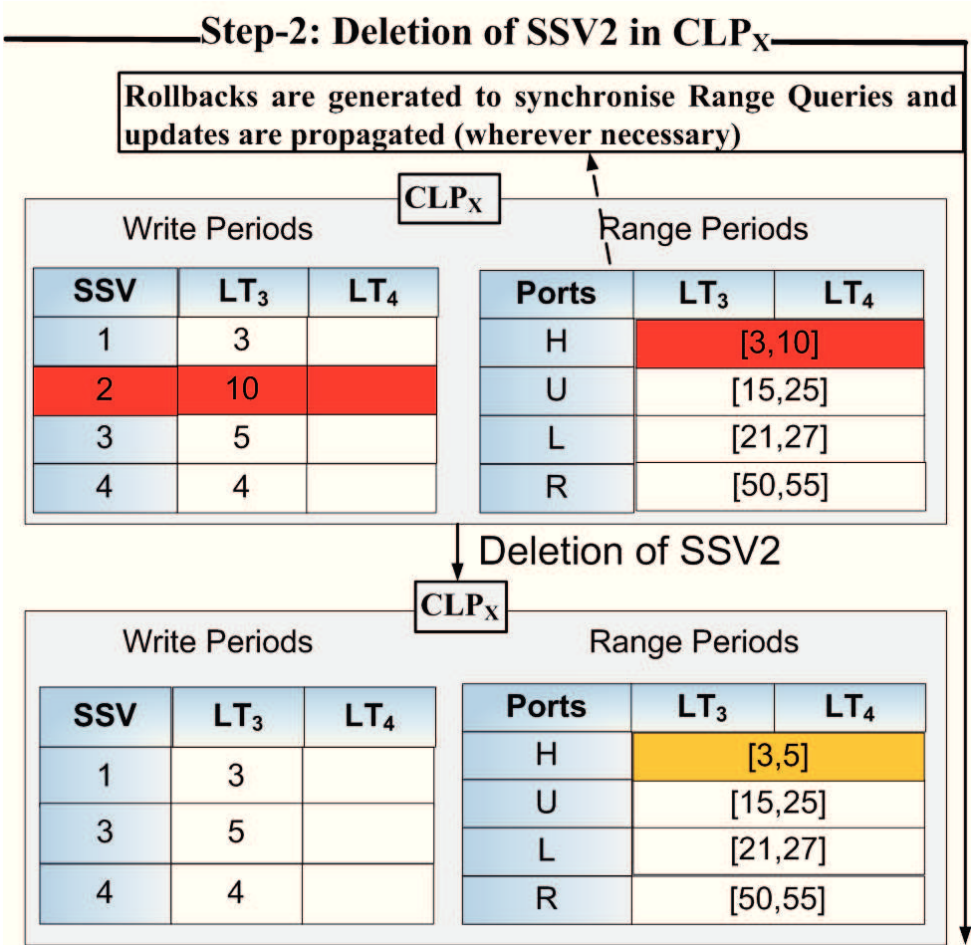


Figure 5.6: State changes in Write Periods and Range Periods table after deleting $SSV-2$.

queries, one per simulation tick. Range queries are issued with a static position and a radius corresponding to the *RQ size* experimental parameter. Simulations are run for 1000 ticks.

The traces used were generated by varying the two experimental parameters: *SSV range*, and *RQ size*; both proportional to the size of 2-dimensional area covered by the simulation. Figure 4.13 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 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

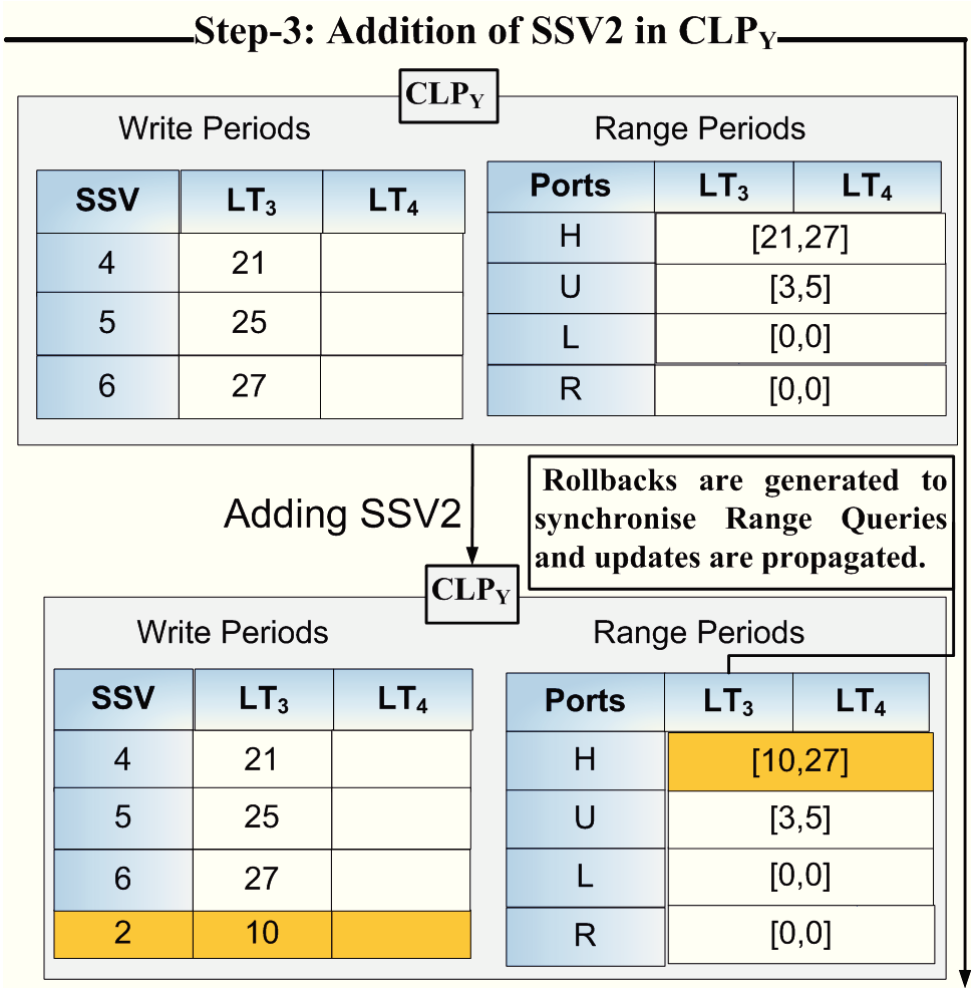


Figure 5.7: State changes in Write Periods and Range Periods table after adding $SSV-2$.

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 observed. Note that for both parameters a phase transition can be observed 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.

In total $5 \cdot 5 = 25$ traces were generated for 5 values for both *SSV range* and *RQ size* (0.1, 0.2, 0.3, 0.4, and 0.5 proportions of the simulation area considered), and 3 experiments were done for each trace with different pseudo random number generated seed values for $3 \cdot 25 = 75$ experiments in total.

5.2.2 Range Query Propagation

In the first 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*.

Figure 5.8a 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

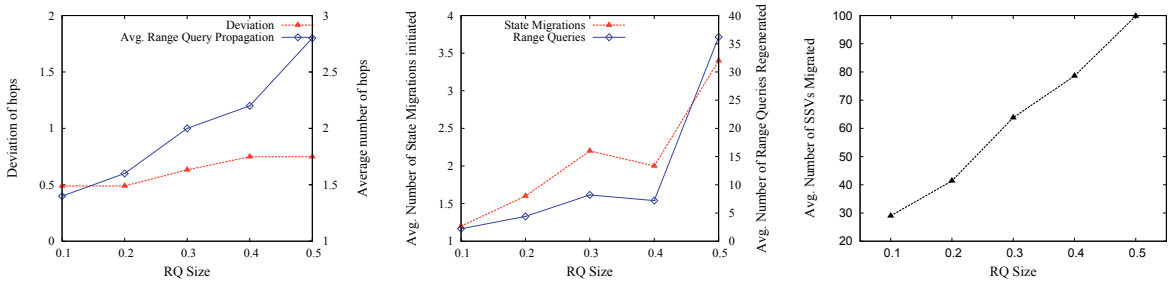


Figure 5.8: Average number of hops (with its deviation from the mean), average number of State Migrations initiated and Range Queries regenerated for varying *RQ size* and average number of SSVs migrated per state migration.

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, number of SSVs migrated per state migration 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 5.8b 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 observance of a non-linear trend (when *RQ size* increases from 0.3 to 0.4) is because of overlapping access patterns migrating larger number of SSVs with few state migrations. This does not alter the fact that the number of SSVs migrated in total increases with *RQ size* values as evident in figure 5.8c. The average number of Range Queries regenerated follows the trend presented in figure 5.8b 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.

Thus far, 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:

$$C_r = \frac{C_{-SM} - C_{+SM}}{C_{-SM}} \cdot 100 \tag{5.1}$$

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.

Table 5.3 shows the *Cost Reduction* for different *SSV range* and *RQ size* values. The

RQ size	***SSV range***				
	0.1	0.2	0.3	0.4	0.5
0.1	0	0	69.42	67.38	49.82
	(0)	(0)	(3.86)	(1.15)	(13.48)
0.2	0	67.51	67.32	59.27	47.49
	(0)	(1.24)	(0.09)	(5.65)	(13.98)
0.3	67.19	67.19	57.09	49.89	45.85
	(0.75)	(0.05)	(7.1)	(5.73)	(7.01)
0.4	67.06	56.96	47.02	47.54	33.01
	(2.12)	(6.07)	(0.26)	(24.28)	(1.55)
0.5	43.53	41.39	48.61	46.47	31.2
	(1.67)	(0.02)	(16.19)	(13.38)	(1.26)

Table 5.3: Cost reduction C_r (standard deviation) in percentages for different *SSV range* and *RQ size* combinations.

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.

Table 5.3 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 5.9.

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

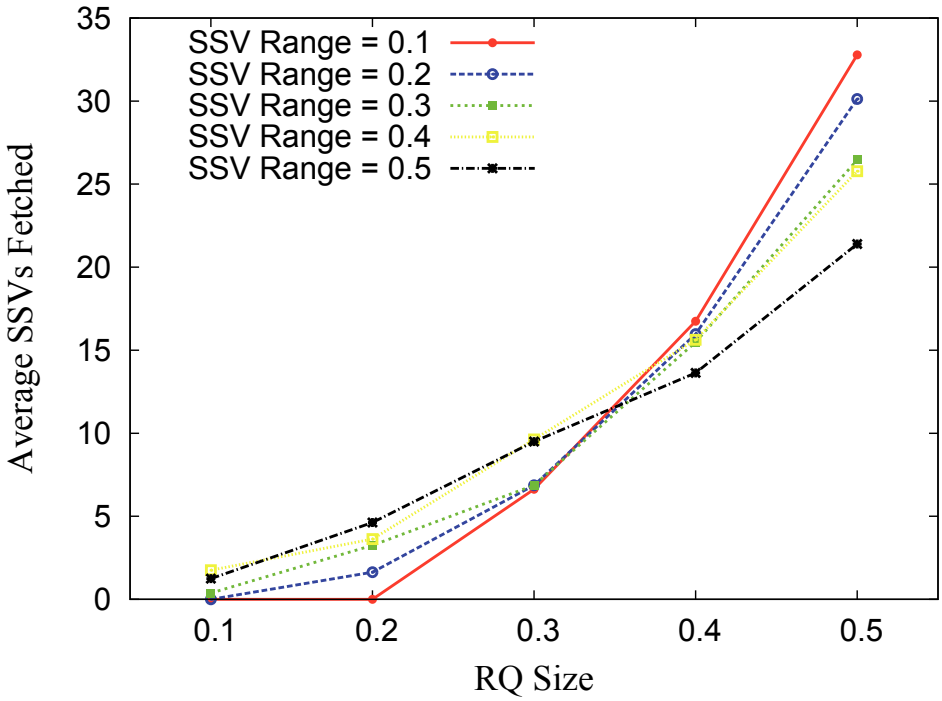


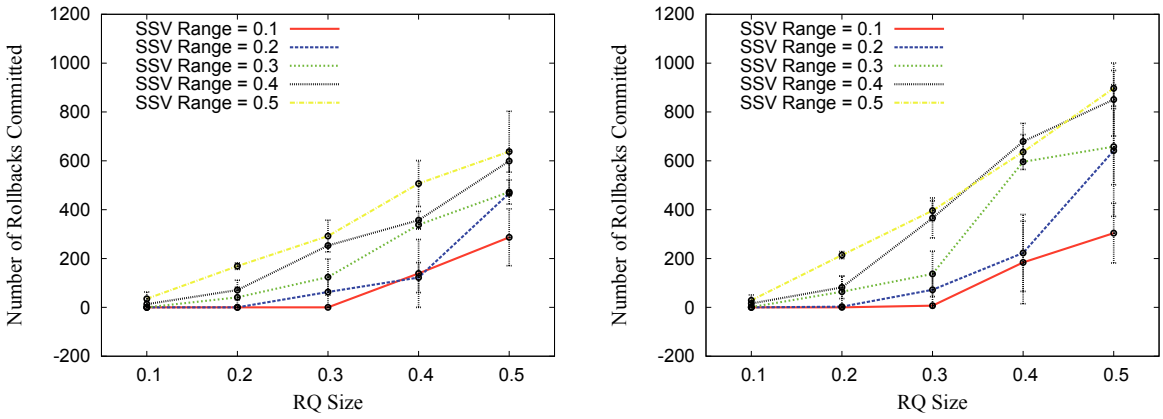
Figure 5.9: Average SSVs fetched by a Range Query for different *SSV range* and *RQ size* combinations

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.

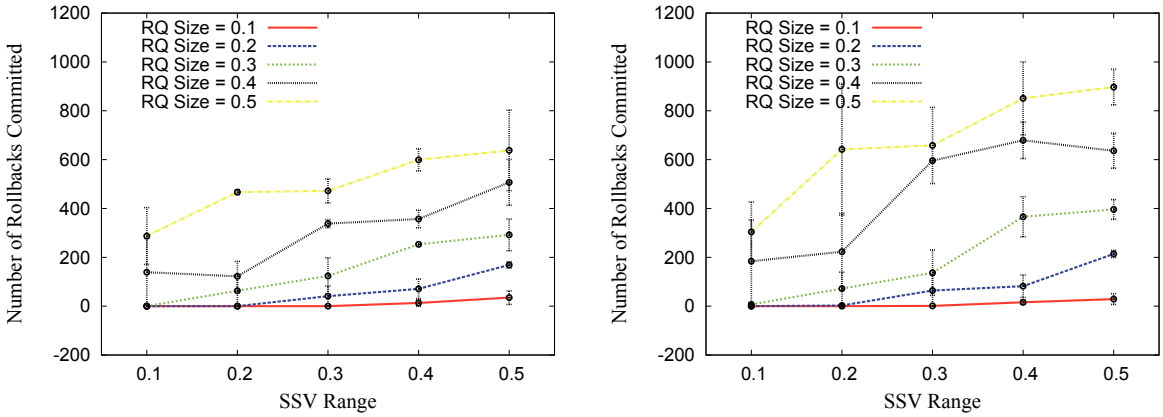
5.2.3 Rollback volatility

Thus far 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.



(a) Avg. Rollbacks Committed without State Migra-(b) Avg. Rollbacks Committed with State Migration tion



(c) Avg. Rollbacks Committed without State Migra-(d) Avg. Rollbacks Committed with State Migration tion

Figure 5.10: Rollbacks committed per ALP for varying *RQ size* and *SSV range* with their standard deviations.

Figure 5.10 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 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

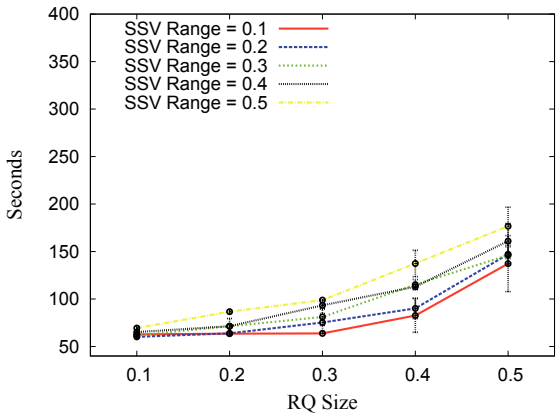
was varied plotted for different *RQ size* parameter values (with and Without SM). As the results were averaged over 3 runs with different pseudo random number generator seeds, the standard deviation for each result in the graphs is depicted using error-bars.

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 [79, 74]. 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.

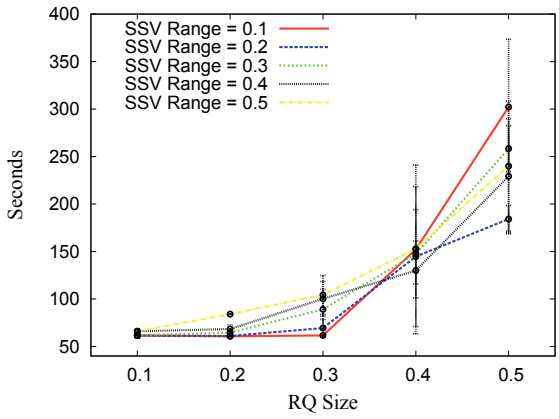
The result presented in figure 5.10 bear out this expectation. The number of rollbacks increase almost linearly with larger *SSV range* and *RQ size* parameter values. With *RQ size* 0.1, the number of committed rollbacks is close to 0, both with and without state migration. Without state migration, 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. In general, as expected, the number of rollbacks committed with State Migration is higher than with state migration.

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 5.11 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.

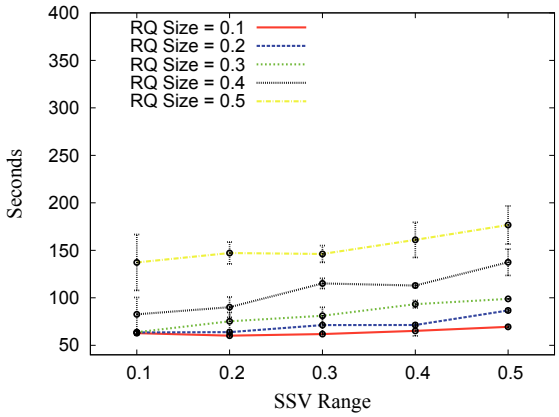
Figure 5.11 suggests that the average simulation time increases with increases in both the *SSV range* and *RQ size* parameters. Without state migration the average simulation time ranges from 60 seconds for *SSV range* 0.2 and *RQ size* 0.2 to 176 seconds for *SSV*



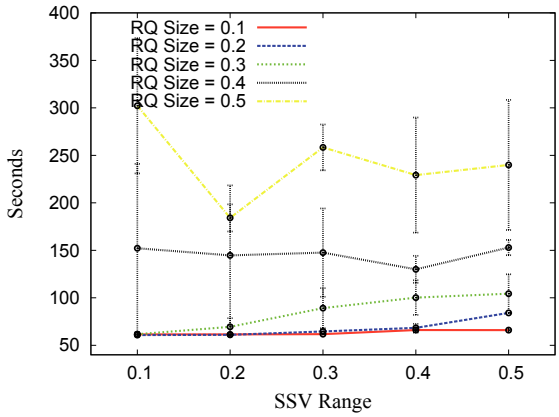
(a) Average Simulation Time without State Migration



(b) Average Simulation Time with State Migration



(c) Average Simulation Time without State Migration



(d) Average Simulation Time with State Migration

Figure 5.11: Average Simulation Time for varying *RQ size* and *SSV range* with their standard deviations.

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 5.10.

With state migration the average simulation time increases linearly upto *RQ size* 0.3, increasing exponentially beyond that. 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 5.10. With the query propagation trend with state migration as shown in figure 5.8a 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:

1. 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 5.8a), 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 increase 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.

   Figure 5.12 illustrates this pattern by showing *Query Response Time* during a typical simulation run both with and without state migration. As expected, figure 5.12 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

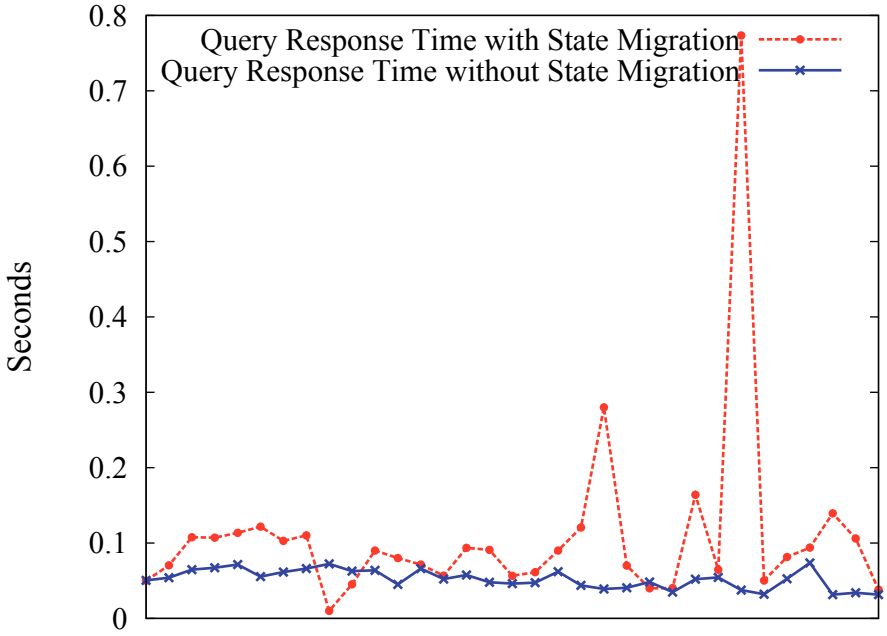


Figure 5.12: Query Response Time with and without State Migration for a typical simulation run.

over the run *Query Response Time* is negatively affected.

2. 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 5.8a) 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*. Similar effect can be observed with smaller values of *SSV range* while the size of a range query is increasing.

   This effect is illustrated by the average rollback depths found for different *SSV range* and *RQ size* parameter values as shown in table 5.4. Although no linear trend can be distinguished for all *SSV range* and *RQ size* combinations, we note that for *RQ*

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

size 0.1 this depth is almost 10 whereas for *RQ size* 0.5 it is just 4.3. The table also bears out our expectation that the rollback depth is quite high with smaller values of *SSV range* and *RQ size* (as we can observe the values from top half along the diagonal). It also suggests that as the *RQ size* increases with *SSV range* values, 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 is fewer (see figure 5.10), the depth of these rollbacks is higher, making them more expensive to perform and thus negatively affecting simulation time.

In summary we conclude that the inclusion of state migration has a mixed effect on simulation time. For small *SSV range* and *RQ size* inclusion of state migration results to a clear decrease in the simulation time. In this range state migration moves the SSVs close enough to the accessing ALPs for the decreased access cost of these SSVs to outweight the extra costs incurred through the need for more rollbacks and updates necessitated by state migration. Beyond these *SSV range* and *RQ size* values the inclusion of state migration yields an increased overhead. This is caused primarily by the extra number of rollbacks needed for state migration while when the number of rollbacks needed is not increased, performing these rollbacks is more expensive.

5.3 Summary

In summary, to achieve scalability in a distributed simulation, the framework should be dynamic and adaptable. Within this framework, the goal is to reduce the horizon of a range query based on the access patterns of the agents in the simulation. *State Migration* algorithm migrates a set of SSVs towards the agents that access the most. The state migration reduces the propagation extent of the range queries but introduces extra overhead in the system. Evaluation of the proposed approach in section 5.2 has shown a significant reduction in the access latency and cost of the simulation for varying *RQ size* and *SSV range* parameter values. The volatility of rollbacks has a direct relation with varying *RQ size* and *SSV range* parameter values. The volatility of rollbacks has an adverse effect initiating state migration with minimum gain. It also shows that the extra cost of migration process does not necessarily compromise the performance of the system. Though the results are not promising it has been tested with traces generated in random using simulation toolkits. These random traces have turned out be a worst case scenario. In the following section, PDES-MAS kernel integrated with all the modules is tested with real MAS to test its scalability and efficiency of data access and migration algorithms.

CHAPTER 6

EXPERIMENTS

This chapter presents an evaluation of PDES-MAS system integrated with algorithms presented in chapters 4 and 5.The experimental analysis is carried in two parts: first, as a benchmark section 6.2 presents an investigation into analysing the performance of PDES-MAS simulation kernel using traces generated from MAS simulation test beds such as TileWorld[111] and Boids[119]. The aim of this investigation is to analyse the impact of locality based access patterns on the system measured with different performance metrics. Second, a scalability analysis under different conditions and CLP-tree configuration is presented in section 6.3. For this purpose, we have used two different computational infrastructures such as **Birmingham eScience cluster** [1] and **Bluebear Cluster** [2].

6.1 Integrated PDES-MAS kernel

The algorithms presented in chapters 4 and 5 are built within the assumption and philosophy of PDES-MAS simulation kernel. The architecture of the PDES-MAS framework is depicted in figure 6.1. Within figure 6.1, modules handling algorithms of time synchronised range queries and state migration are depicted as blocks with crossed lines.

Basically, the figure depicts a flow of different messages (range query, ID query/write,

[1] A detailed overview of hardware and user information can be found at `http://www.ep.ph.bham.ac.uk/general/escience-cluster/`

[2] Architecture and user information of the cluster can be found at `http://www.bear.bham.ac.uk/bluebear/`

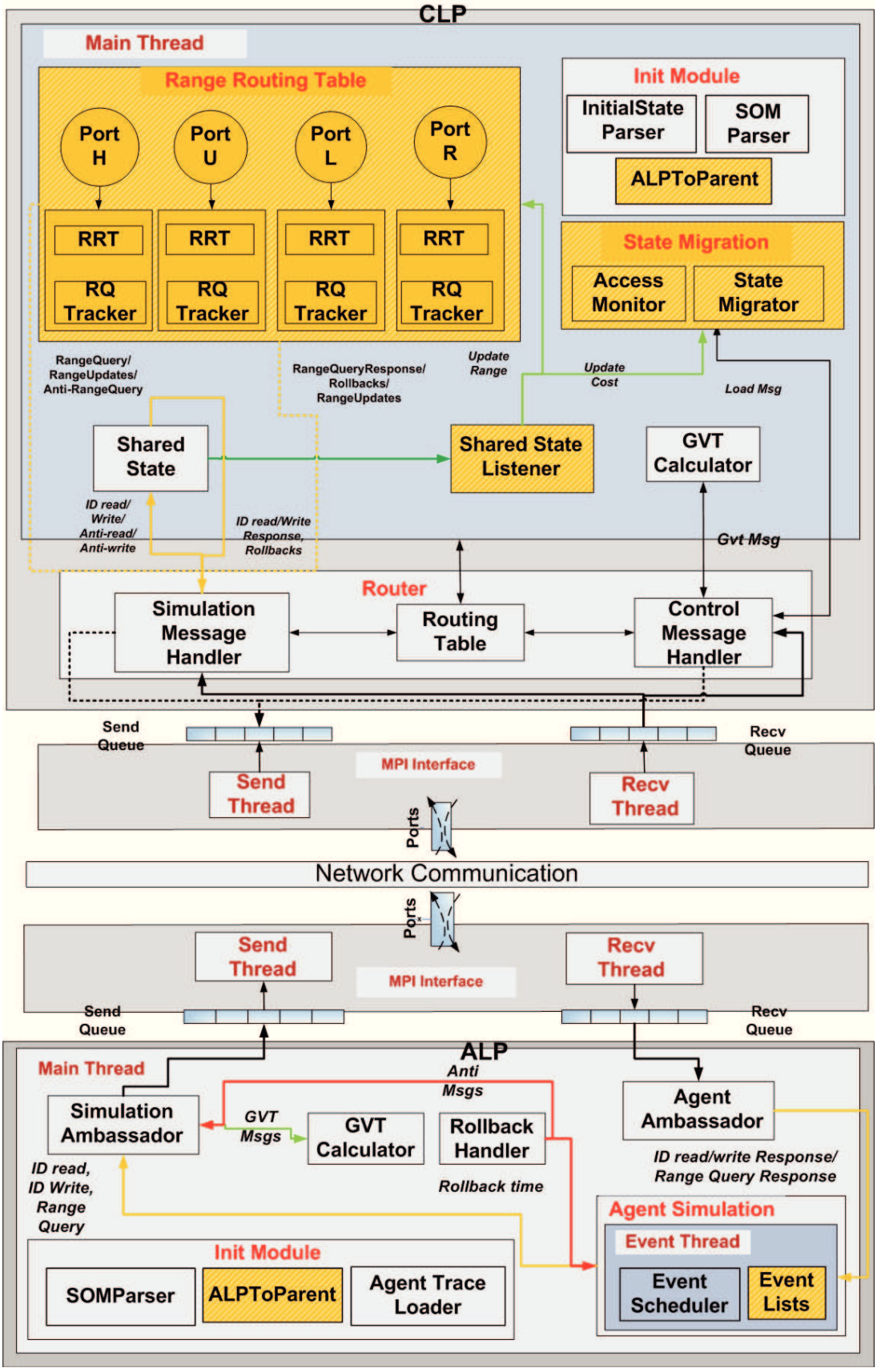


Figure 6.1: PDES-MAS architecture.

rollbacks) and threading mechanisms within an ALP and a CLP. Algorithms of time synchronised range queries are handled with *RangeRoutingTable* and *Range Periods* modules, whereas monitoring and distribution of shared state are handled with two modules namely *Access Monitor* and *State Migrator*. Obviously, the purpose of access monitor is to maintain statistics containing time and frequency of ALPs access on SSVs at each port. *State Migrator* initiates migration procedure and updates migrating SSVs' new destination port. A summary of implementation changes in the PDES-MAS framework are as follows,

1. Locations of ALPs in the tree are identified with a trace file having tuples of the format $(Alpid, Clpid)$. This association would enable attaching any number of ALPs to a CLP (instead of a strict binary tree representation). In the trace file, each ALP id is associated with a CLP id (parentCLP) to which it is attached.

2. Multiple event queues to hold traces of multiple agents in an ALP (instead of strict association of $1agent/ALP$). For that purpose, each agent maintains its own LVT. Within an ALP, the event scheduler schedules events of multiple agents in a round robin fashion. A rollback for an agent would not affect the progress of the other agents in an ALP.

3. *Shared State Listener* module is implemented based on an *Observer design pattern* in which all event actions on shared state module such as ID Read, Write and Range Query are disseminated automatically to all listening modules such as *Access Cost Monitor* and *Range Routing Table* modules.

In the following sections, an evaluation of the PDES-MAS system under different conditions and CLP-tree configurations is presented. The results obtained from the analysis is expected to provide an insight to the behaviour of the algorithms in response to the experimental parameter values of agent based simulation models. A similar analysis presented in chapters 4 and 5 are evaluated with synthetic traces generated with a uniform distribution of virtual locations of agents and shared state variables. In the following sections, the simulation models are chosen with agents exhibiting a notion of locality based

access patterns. However, what remains to be seen how the access patterns (controlled with experimental parameters) influence the performance of the system.

6.2 Benchmark Analysis

In this section we present a benchmark analysis of the system with a number of experiments using traces generated from agent based simulations such as TileWorld [111] and Boids [119]. TileWorld [111] and Boids [119] are two well known test beds in the MAS community. The experimental investigation is carried out to understand and analyse the impact of locality based access patterns on the performance of the system. Tileworld is developed using *GridWorld* [49, 146] simulation toolkit [3]. The tool kit is a part of case study to develop agent based modeling systems. The toolkit provides a graphical environment where visual objects interact and inhabit in a two-dimensional grid. It also provides a mapping of transforming objects actions on the environment to a GUI visualisation. Boids is developed within the implementation of MWGrid multi-agent simulation toolkit under development at the University of Birmingham [4]. MWGrid [141, 24, 100] explores the military logistics of battle of Mazikert in 1071 using agent based modeling and distributed simulations (a detailed overview is presented in chapter 7).

The experiments in this section have been carried out in Midlands eScience Center (MeSC) cluster environment. At the time of experimentation, it consisted of 15 worker nodes each with 2 GBytes of memory and 2 Intel Xeon 3GHz processors. All worker nodes are accessible from the clusterś master node connected together by 100MBps fast ethernet running Red Hat Enterprise Linux AS release 3.2, kernel version 2.4.21. It has been installed with *LAM/MPI (7.1.3)*[131] libraries for experiments to launch and establish communication between processes. The simulation test beds chosen for this investigation would provide a good benchmark for the system as they have completely

[3]The architecture and framework details can be found at (`http://www.collegeboard.com/student/testing/ap/compsci_a/case.html`)

[4](see `http://www.cs.bham.ac.uk/research/projects/mwgrid`) for more details on the developments of the project

different agent behaviours and simulation environments. Analysis on the performance of the system under different conditions using both of these test beds are provided in sections 6.2.1 and 6.2.2.

6.2.1 TileWorld

TileWorld [111] is a well known agent based test bed. It has been used to study agent's reasoning and committing strategies such as changing the existing plan or replan in case of obstacles in the environment[111]. The environment consists of a 2-D grid of squared cells. The objects in the environment are agents, tiles, holes and obstacles such as rocks. The objective of an agent is to score many points by pushing the tiles into the holes. The environment is highly dynamic i.e., tiles, holes and obstacles appear and disappear dynamically based on a user controlled simulation configurations. A snapshot of TileWorld simulation is shown in figure 6.2. Smily faces depict agents, Black squares are Tiles, black irregulars are rocks and blue stripped squares are holes. An agent, at each step, goes through a 'sense-think-act' cycle and as the simulation progresses each agent goes through several of these cycles. In essence, an agent senses the environment, prioritise the selected targets (if there is more than one target within sensor range) and decides to move towards a selected target using A* route planning with a minimum cost. The cost, here, is the number of cells required to reach a target. To avoid computation deadlock, the number of iterations to calculate a best route to a selected target is set to 128. A summary of a set of basic Tileworld simulation rules are

- Agents can move to any square in the Grid which does not contain an obstacle.
- When it stands on the same square as the Tile, it can pick up and increment the number of tiles it carry.
- When it stands on the same square as the Hole, it can drop tiles to the hole based on the depth. If it fills the depth of the hole, it can take number of points as hole depth.

- New Tiles, holes and obstacles are generated based on some parameterised frequency.
- Agent's priority is to score as many points as possible.
- It generates a plan to select a target that provides more reward. Replanning happens when obstacles appear dynamically on its path.

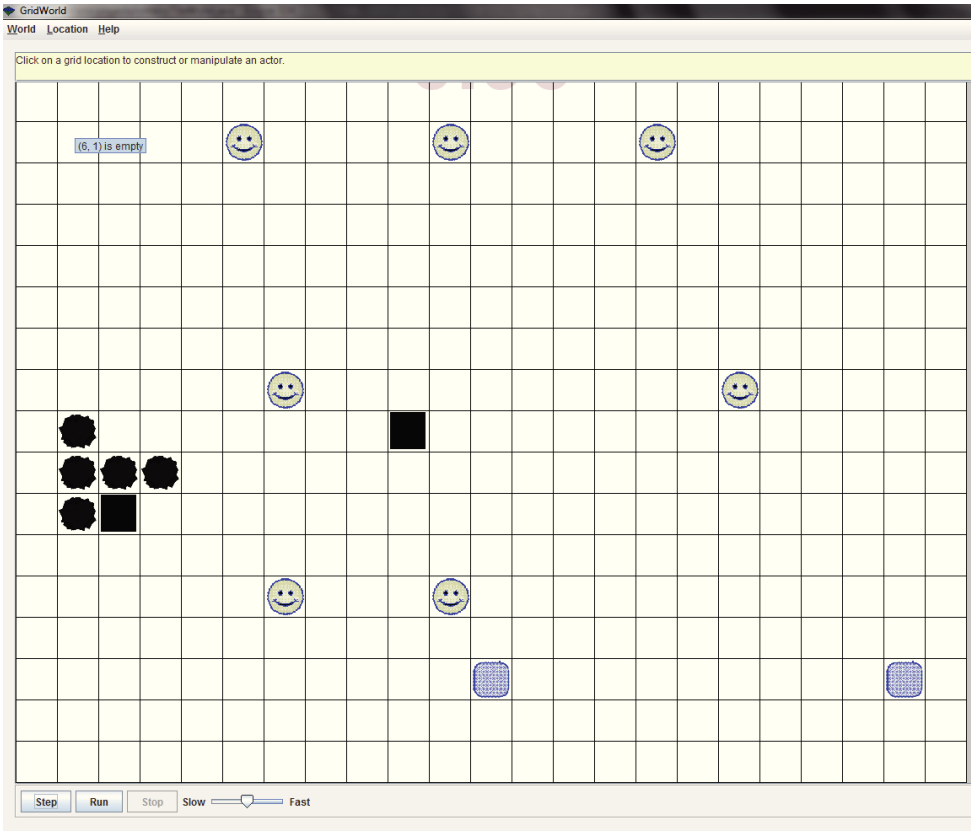


Figure 6.2: A snapshot of TileWorld Simulation Test bed. Smily faces depict agents, Black squares are Tiles, black irregulars are rocks and blue stripped squares are holes.

The simulation is developed within the implementation of *GridWorld*[49, 146] simulation toolkit. Sensing the environment within a sensor range (range configured by programmer) gets back with a list of objects within that range. Here, a single range covers **8 adjacent cells** around the agent, but it can move only in four directions *(UP, LEFT, RIGHT, DOWN)*. At any time an agent with sensory range r can cover **8(1+2+...+r) cells**. A worthy target is selected based on its reward (points) and the agent's priority. An agent on selecting a worthy target uses the target location to calculate a best route

with A^* *planning* algorithm. It then generates a list of actions to be executed to reach the target. At every step, before executing any action, it has to make sure whether the action is still valid or not, for example, if the action is to move to a cell location $(2, 5)$, but cell $(2, 5)$ (previously empty) is now occupied by an obstacle, then agent has to replan.

When there is no specific target, an agent just moves to a random adjacent location. In essence, the simulation generates number of actions that are in sequence and interacts with the environment dynamically. The agents, here, are completely autonomous and unpredictable such that their access patterns are bit random because they can change their courses at any point of time in the simulation. It is ideal for the investigation as it is dynamic and yet agents go through a ('sense, think and act') life cycle. The experimental setup and parameters are presented in the following section.

6.2.1.1 Experiment setup

This section presents the experimental setup and simulation parameters of TileWorld simulation. Tileworld is highly configurable simulation system but for investigation the parameters used for our investigation are presented below,

1. **Environment Size** - The size is varied i.e., number of rows and columns in the grid is varied.

2. **Sensor Range** - This is the visibility of the agent in the environment.

3. **Creation Probability** - Dynamically objects are created during the course of the simulation based on this configuration. More objects are created if the probability it high.

4. **Object life time** - The life time of the created object before it is removed from the environment.

5. **Number of Agents** - The number of agents interacting in the environment.

The table 6.1 presents the simulation parameters of TileWorld simulation under investigation.

Table 6.1: Simulation Parameters

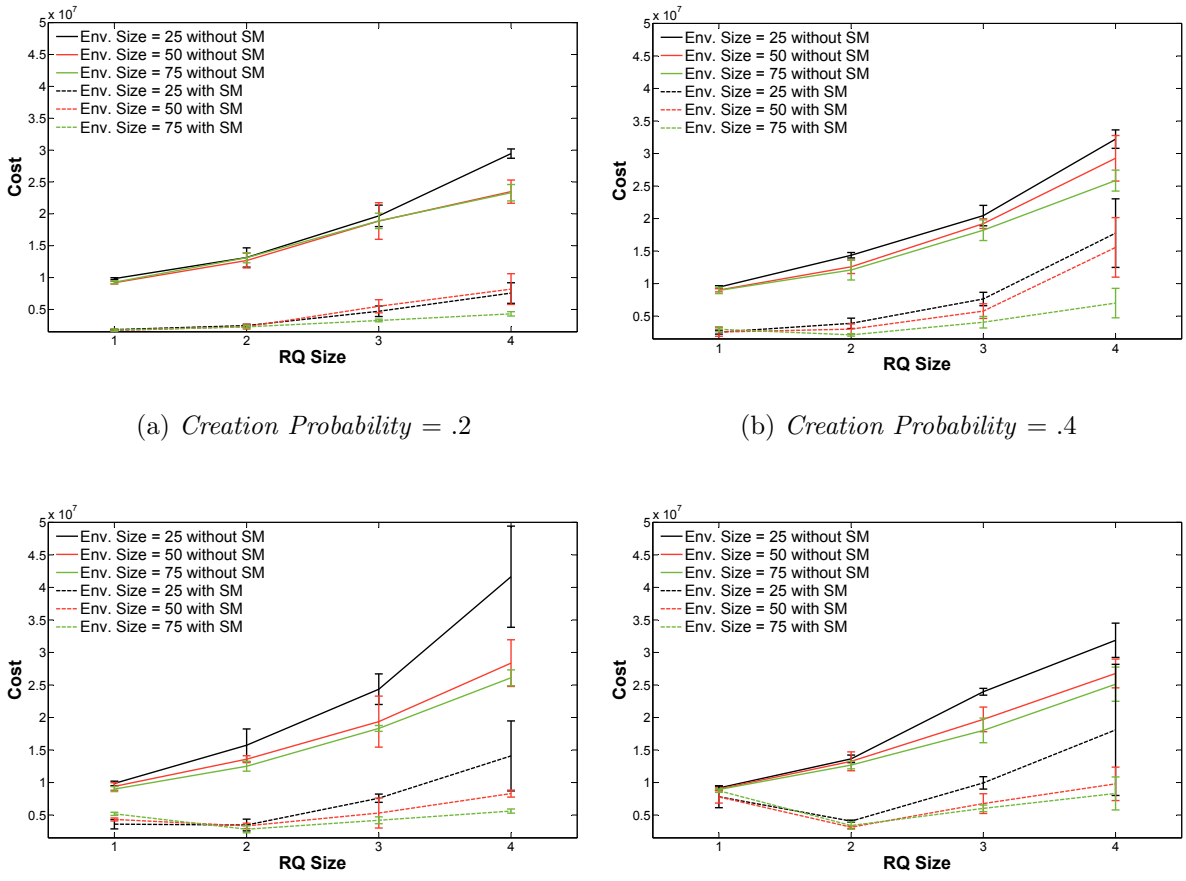
Environment Size	30 x 30, 45 x 45, 65 x 65
Range Query Size	1, 2, 3, 4
Creation Probability	.2, .4, .6, .8
Object life time	50
Number of Agents	16

The experimental setup for this analysis consisted of a CLP-tree of 7 CLPs. The objects (Tiles, holes, rocks and agents themselves) are modeled as SSVs. All SSVs were placed at the root CLP to more clearly show the effect State Migration has on performance. 16 ALPs, four attached to each leaf CLP. Simulations are run for 300 logical time ticks (or steps). There are 3 random seeds used for each simulation parameter, so in effect 3.(3.4.4) = 144 experiments (where each experiment is run for 20 times) were run for this simulation analysis. The investigation focuses on the influence of three simulation parameters such as *RQ size*, *Creation Probability* and *Environment Size* in the system measured by three metrics: *Access Cost, Range Query Propagation and Simulation Time.* Results for each of these performance metrics is presented in the following sections.

6.2.1.2 Access Cost and Query/Update Propagation

This section presents the impact of accessing SSVs under different conditions of experimental parameters of TileWorld simulation. The performance is measured by two metrics: Access Cost and Query/Update propagation. The total access cost of the simulation is measured by the sum of the access cost of all SSVs. The access cost of a SSV is the product of the number of accesses with number of hops required to fetch that SSV. The figure 6.3 presents the total access cost of the simulation for varying *RQ size*, *Creation Probability* and *Environment Size* values.As the results were averaged over different experimental

runs including 3 different pseudo random number generator seeds and 20 runs of each seed), the standard deviation for each result in the graphs is depicted using error-bars. The error bars are indicators to the variance in the values of access cost over different experimental runs. For comparisons, access costs with and without SM (State Migration) are presented.



(a) *Creation Probability* = .2

(b) *Creation Probability* = .4

(c) *Creation Probability* = .6

(d) *Creation Probability* = .8

Figure 6.3: Access Cost for varying *RQ size* and *Creation Probability* values with their standard deviations.

The figure 6.3 clearly presents the impact of all experimental parameters with and without SM. The cost increases with the increase in *RQ size* and *Creation Probability* parameter values. The graphs do not illustrate a complete linearity but bears out some inferences of the impact of experimental values in the system. As the agents range window (sensor range) increases with *RQ size* parameter, it increases the possibility of SSV

accesses. Also the likelihood of finding SSVs in the environment increases with *Creation Probability* parameter. We can also observe the impact of combinations of both *RQ size* and *Creation Probability* parameters on access cost With SM. When *RQ size* and *Creation Probability* values are set to minimum, minimal number of SSVs accessed and mostly agents update themselves (as they move around). So, With SM cost is reduced significantly with localised SSV access. But as *Creation Probability* values increase (keeping *RQ size* values minimum), the number of SSVs accessed increases with the cost as the range window is not large enough to localise SSV access. As both parameters are increased linearly, the effect is diminished linearly. However, with the combination of *Environment Size* parameter values, the cost overall seems to decrease with increasing environment size as it reduces the likelihood of exploring SSVs in the environment. This is observed from the figure 6.3 as Without SM, the cost varies from 8922913 (with *RQ size* = 1, *Creation Probability* = 0.8 and *Environment Size* = 65) to 41648342 (with *RQ size* = 4, *Creation Probability* = 0.6 and *Environment Size* = 30) and With SM, it varies from 1693118 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 45) to 18089680 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30). With SM, the maximum gain or reduction in cost is 27554914 (with *RQ size* = 4, *Creation Probability* = 0.6 and *Environment Size* = 30) and minimum gain or reduction in cost is 180343 (with *RQ size* = 1, *Creation Probability* = 0.8 and *Environment Size* = 30).

To show more clearly the impact of three experimental parameters (as mentioned above) we have compared the access cost of the simulation without and with state migration by calculating the difference ratio in percentages called the *Cost Reduction*: C_r (similar to equation 5.2.2 presented in section 5.2 within chapter 5). *Cost Reduction* in percentages for varying *RQ size*, *Creation Probability* and *Environment Size* values is presented in table 6.2.

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	81.36	73.19	63.25	14.07
2	81.11	72.94	78.26	70.17
3	76.05	62.35	68.44	58.56
4	74.3	45.26	65.96	40.12

(a) *Environment Size* = 30

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	81.58	71.72	53.93	13.41
2	81.99	76.39	75.34	75.95
3	70.34	69.73	73.76	65.78
4	64.22	47.8	70.06	62.77

(b) *Environment Size* = 45

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	81.06	66.35	42.14	2.02
2	82.58	82.44	77.53	73.1
3	82.68	77.36	77.03	66.43
4	81.41	72.77	78.51	66.32

(c) *Environment Size* = 65

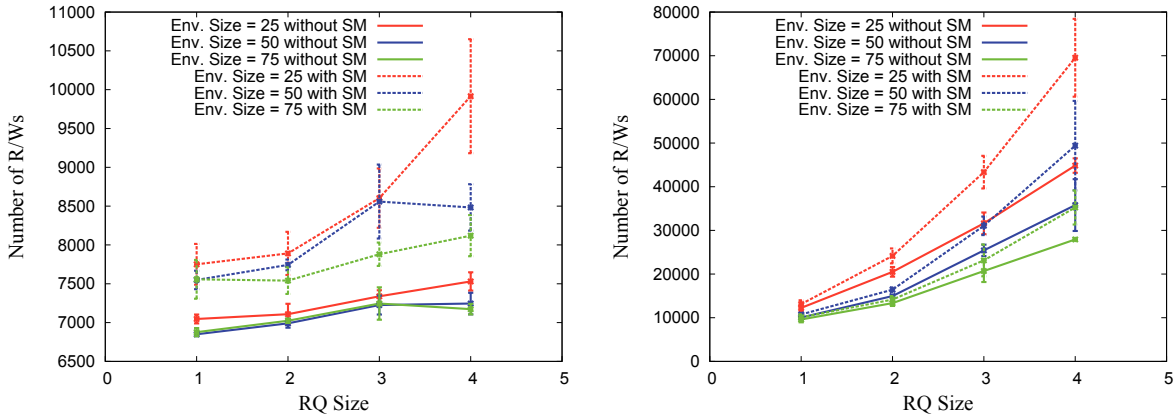
Table 6.2: *Cost Reduction* in % for varying *RQ size*, *Creation Probability* and *Environment Size* values of TileWorld Simulation.

The table bears out the expectation that with minimum range window, increasing *Creation Probability* values minimise *Cost Reduction* (or increases cost with minimal local access) as the cost reduces from 81.36 % (with *RQ size* = 1, *Creation Probability* = 0.2, *Environment Size* = 30) to just 14% (with *RQ size* = 1, *Creation Probability* = 0.8, *Environment Size* = 30). But with increasing range window, SSV migrations increases to localise SSV access thereby reducing cost from 78.2% (with *RQ size* = 4, *Creation Probability* = 0.2, *Environment Size* = 30) to 45.73 % (with *RQ size* = 4, *Creation Probability* = 0.8, *Environment Size* = 30). We can also observe that *Cost Reduction* increases from 64.09% (with *Environment Size* = 30) to 69.36% (with *Environment Size* = 65). This pattern is observed for all values of *Environment Size*. Overall gain in cost for TileWorld simulation is 66.46% for TileWorld simulation.

The graphs presented in figure 6.3 also illustrate that the variance in the cost increasing with the increase in experimental values. The error bars, in the graphs, are presented in view of illustrating the impact of rollbacks on the variance of the cost on several experimental runs of the same experiment with different random seeds. The variance in the cost is quite high with increasing *RQ size* and *SSV range* values because of the overlapping access patterns increasing the probability of rollbacks (increasing the number of regenerations of accesses such as reads and writes). The variance of the cost Without SM ranges from 142139 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 30) to 2635622 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30) and With SM it ranges from 142139 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 30) to 2635622 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30).

The cost presented in figure 6.3 With SM includes the extra cost incurred with the initiation of state migration by sending rollbacks to agents in addition to rollbacks created non-deterministically by straggler writes. To quantify the effect, the figure 6.4 presents the number of accesses (reads and writes) generated by agents for varying *RQ size* and *Creation Probability* values. Without SM, the number of SSVs accessed varies from 6849 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 45) to 9267.33 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30) and With SM, it varies from 7532.33 (with *RQ size* = 2, *Creation Probability* = 0.4 and *Environment Size* = 65) to 15252.33 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30).

On average the number of accesses With SM increases by 20% (with *Environment Size* = 30), 14%(with *Environment Size* = 45) and 9%(with *Environment Size* = 65). We can also observe the number is almost halved on increasing the size of the environment as the number of migrations required to localise data access is minimum. We can also observe the variance in the number of accesses is clearly inline with the pattern of cost presented in figure 6.3. However, the extra cost is compensated on reducing the propagation of SSV



(a) *Creation Probability* = .2

(b) *Creation Probability* = .4

(c) *Creation Probability* = .6

(d) *Creation Probability* = .8

Figure 6.4: Number of Reads/Writes generated for varying *RQ size* and *Creation Probability* with their standard deviations.

access. Propagation is calculated by number of hops required for a read/write to parse through the CLP tree and get back with a result. The SSVs are placed at root CLP to clearly show the effects of migration. Without SM, the number of hops is constant for both Range Query and Update propagation, which is 7. 3 hops from the leaf CLP to the root, 3 return hops, and an extra hop to the ALP. The same method is used for all experiments. But the expectation is that migration moves SSVs closer to the accessing agents reducing the propagation. Range Query (read) and update (write) propagation With SM is presented in tables 6.3 and 6.4.

The table 6.3 shows an average reduction in range query hops. In tileworld simulation, the area polled by agents will change constantly (almost every step) and so forth reflects in

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	5	5.33	5.67	5.33
2	5.67	5.33	5.33	5.67
3	5.33	5.67	5	5.67
4	5.33	5.67	5.33	7.33

(a) *Environment Size* = 30

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	5.33	5.33	5.67	5.33
2	5.33	5	5.67	5
3	5.33	5	5	5
4	5.33	4.33	5.67	5.67

(b) *Environment Size* = 45

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	5	5	5	5.67
2	5.33	5.67	5.33	5.67
3	5.33	5.33	5.33	7
4	5.33	5.33	5.67	5.67

(c) *Environment Size* = 65

Table 6.3: Number of Range Query Hops for varying *RQ size*, *Creation Probability* and *Environment Size* values.

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	2.67	4	4.67	6.67
2	2	2	3	3
3	1	2	2	2.33
4	1	2	2	2.33

(a) *Environment Size* = 30

RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	2.67	3.67	5	6.33
2	2	2.67	3	3.33
3	1.33	2	2.67	3
4	1.33	2	2	2.33

(b) *Environment Size* = 45

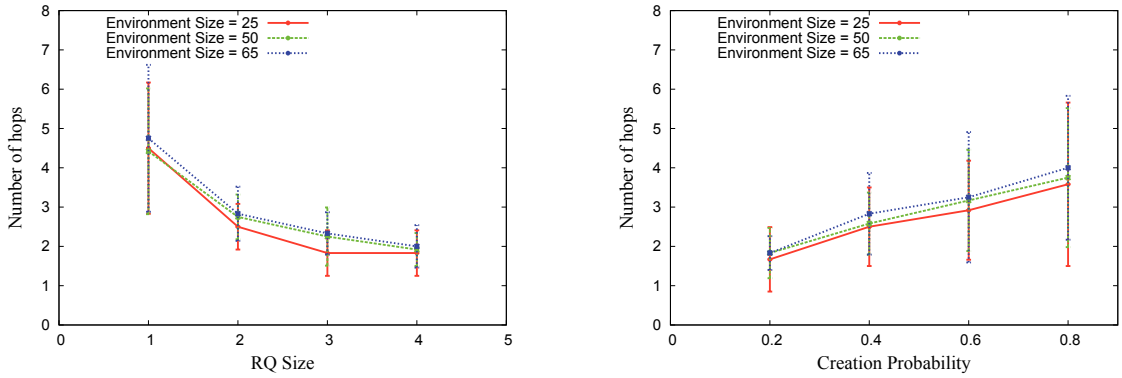
RQ size	***Creation Probability***			
	0.2	0.4	0.6	0.8
1	2.33	4.33	5.67	6.67
2	2	2.67	3	3.67
3	1.67	2.33	2.33	3
4	1.33	2	2	2.67

(c) *Environment Size* = 65

Table 6.4: Number of Update hops for varying *RQ size*, *Creation Probability* and *Environment Size* values.

their propagation. With SM, overall range query propagation is reduced to 5.4. However, table 6.4 clearly presents the distribution of SSVs around the CLP tree. As agents update themselves and SSVs within their sensor range (more often), on average it is inclined towards more localised access to 2.8 (compared with 7 without SM). This is shown more

clearly in figure 6.5a which presents an average update propagation for varying values of *RQ size* and *Environment Size* and variance over different *Creation Probability* values. It shows that increasing the range window decreases the propagation with its deviation. With *Environment Size* = 30, the hops varies from 4.5 to 1.83 and its variance from 1.67 to 0.58 and the same pattern is observed over different values of *Environment Size*. This is inline with the expectation that with minimum range window (sensor range) and increasing *Creation Probability* value increases the number of hops (as window is not large enough to localise data access). But as range window increases, this effect is diminished with the SSV migrations.



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

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

Figure 6.5

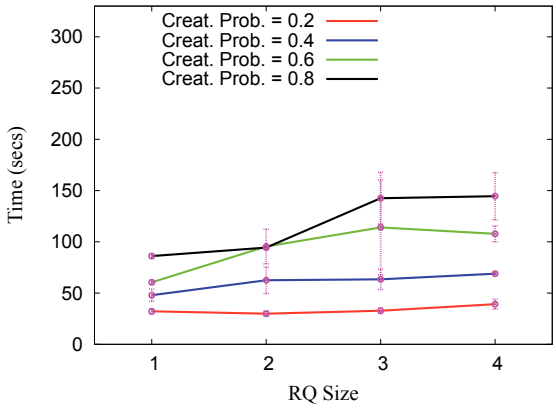
Also it shows that the number increases with increasing *Environment Size* as SSVs are scattered around (reducing the likelihood of SSV access). On the contrary side, figure 6.5b present an average update propagation for varying *Creation Probability* and *Environment Size* values and variance over different values of *RQ size*. It shows that increasing the probability increases the propagation with its deviation. With *Environment Size* = 30, the hops varies from 1.67 to 3.58 and its variance from 0.82 to 2.08 and the same pattern is observed over different values of *Environment Size*.

The results presented so far have measured the access cost and query/update propaga-

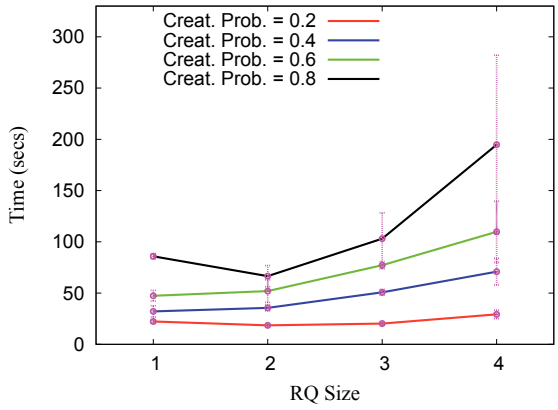
tion for TileWorld simulation. It shows that overall reduction in cost for varying different values of *RQ size*, *Creation Probability* and *Environment Size* is 66.46 %. We also have shown that the extra cost incurred with initiating SM is compensated by the reduction in propagation of query/write issued by the agents. So, the performance of the system, in terms of cost and propagation, has improved substantially.

6.2.1.3 Simulation Time

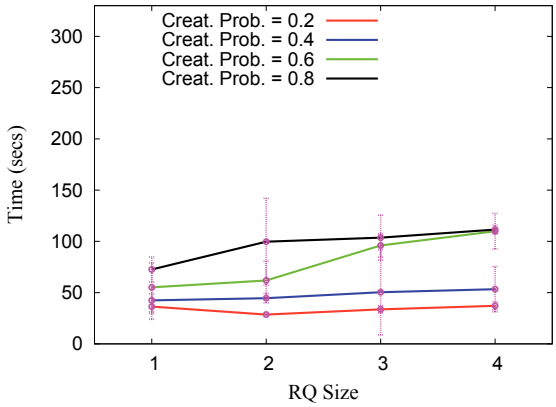
This section presents the impact of initiating SMs on simulation time in the system under different experimental values of TileWorld Simulation. Simulation time is the total elapsed time to finish an experimental run. Thus far observing the patterns of access cost and propagation of SSV accesses in section 6.2.1.2, we would expect a linear relation with the simulation time. This is observed in the figure 6.6 which presents an average simulation time for varying *RQ size*, *Creation Probability* and *Environment Size* values with their standard deviations over different random seeds (including different runs) of experiments. Without SM, the simulation time ranges from 28.07 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 65) to 144.52 (with *RQ size* = 4, *Creation Probability* = 0.2 and *Environment Size* = 30). With SM, it ranges from 17.69 (with *RQ size* = 2, *Creation Probability* = 0.2 and *Environment Size* = 65) to 194.78 (with *RQ size* = 1, *Creation Probability* = 0.8 and *Environment Size* = 30).



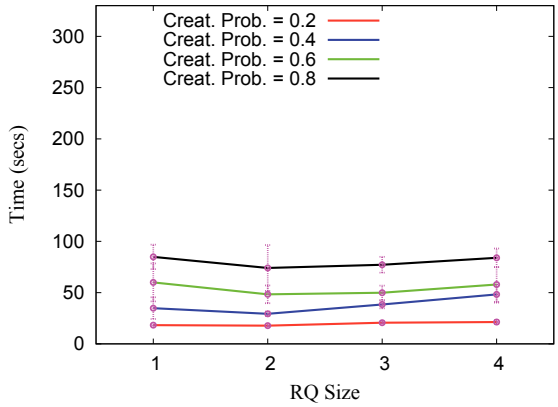
(a) Average Simulation Time without State Migration for *Environment Size* = 25 .



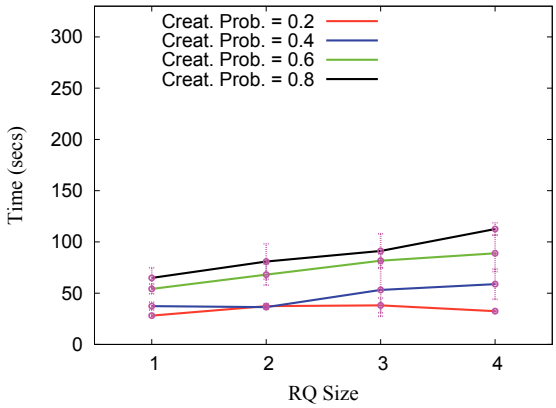
(b) Average Simulation Time with State Migration for *Environment Size* = 25 .



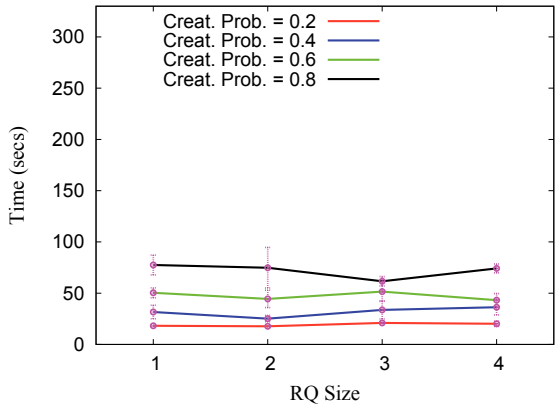
(c) Average Simulation Time without State Migration for *Environment Size* = 50 .



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



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



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

Figure 6.6: Average Simulation Time for varying *RQ size* and *Creation Probability* with their standard deviations.

We can observe that with smaller values of *Environment Size*, increasing *RQ size* and *Creation Probability* increases SSV access with the number of migrations. The cost of migrations (initiating extra rollbacks), though diminished by propagation, increases simulation time to finish the experiments. The variance in the simulation time is also quite high with *Environment Size* values are small. This is because of the imminent threat of the number of rollbacks (due to straggler writes and state migration algorithm). On the other hand, time is not reduced significantly when there are fewer migrations. This is because of the lack of locality based access patterns incurred with smaller visible range. The variance of the time Without SM ranges from 2.33 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 30) to 25.24 (with *RQ size* = 3, *Creation Probability* = 0.8 and *Environment Size* = 30) and With SM it ranges from 1.52 (with *RQ size* = 1, *Creation Probability* = 0.2 and *Environment Size* = 30) to 87.31 (with *RQ size* = 4, *Creation Probability* = 0.8 and *Environment Size* = 30).

This is illustrated with the table 6.5 that presents *Time Reduction* for varying values of *RQ size*, *Creation Probability* and *Environment Size*. We use the similar equation 5.2.2 presented in section 5.2.2 to compare simulation times with and without SM termed as *Time Reduction*.

The maximum reduction in time is 52.67% (observed with *RQ size* = 2, *Creation Probability* = 0.2 and *Environment Size* = 65) and minimum is 0.24% (with *RQ size* = 1, *Creation Probability* = 0.8 and *Environment Size* = 30). We can also observe that as we increase *Environment Size*, the *Time Reduction* increases linearly. Overall *Time Reduction* for TileWorld simulation is 25.94%. To clearly understand the pattern, the table 6.6 presents the number of SMs initiated for varying *RQ size*, *Creation Probability* and *Environment Size*. The table bears out a direct relation to the pattern of *Time Reduction* presented in table 6.5. It shows that the number of SMs initiated increases linearly with increasing *RQ size* and *Creation Probability* values. However, the number of migrations decreases with increase in *Environment Size* as there are few migrations required to localise SSV access. The total number of migrations initiated reduces in total

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	30.83	33.05	21.65	0.24
2	38.27	43.1	45.6	29.5
3	38.32	20.03	32.33	27.66
4	25.43	-2.77	-1.8	-34.78

(a) *Environment Size* = 30

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	49.87	17.85	-8.69	-16.87
2	37.85	34.15	21.87	25.74
3	38.59	23.76	48.05	25.5
4	42.6	9.51	47.25	24.68

(b) *Environment Size* = 45

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	35.21	15.34	7.1	-19.33
2	52.67	30.83	34.74	7.45
3	44.94	36.59	36.93	32.43
4	37.77	38.26	51.46	34.14

(c) *Environment Size* = 65

Table 6.5: *Time Reduction* in percentage (%) for varying *RQ size*, *Creation Probability* and *Environment Size* values of TileWorld Simulation.

from 222.54 (with *Environment Size* = 30) to 66.3 (with *Environment Size* = 65).

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	3.09	0.47	3.68	1.89
2	7.59	4.03	12.75	6.38
3	3.68	17.68	5.19	18.8
4	13.72	11.9	26.84	84.85

(a) *Environment Size* = 30

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	1.7	4.03	2.45	0.94
2	5.72	5.44	7.59	6.79
3	12.04	11.9	21.48	6.53
4	8.99	20.4	9.03	26.88

(b) *Environment Size* = 45

RQ size	*Creation Probability*			
	0.2	0.4	0.6	0.8
1	0.82	0.82	2.49	0.94
2	4.64	0.94	1.25	3.68
3	1.89	2.94	5.91	8.57
4	2.49	3.74	16.51	8.65

(c) *Environment Size* = 65

Table 6.6: Number of State Migrations initiated for varying *RQ size*, *Creation Probability* and *Environment Size* values of TileWorld Simulation.

6.2.1.4 Summary

The results presented so far in sections 6.2.1.3 and 6.2.1.2 attribute to the fact that the performance of the system, in terms of cost, propagation and time, can be reduced With SM. TileWorld is quite dynamic with high volume of SSV access. The system is tested with different experimental values of *RQ size*, *Creation Probability* and *Environment Size* in TileWorld simulation. The impact of different experimental values over cost, propagation and time is presented. It shows that with minimum (optimal) number of migrations, the cost and time can be reduced substantially. Overall gain in cost is 66.46 %, Query propagation on average reduced to 5.4 and update propagation to 2.8 (compared with 7 without SM) and simulation time is reduced by 25.94%.

6.2.2 Boids

In this section we present an analysis of the system with a different agent based modeling simulation called Boids. Boids simulation model is widely used to simulate flocking behaviour of birds or herds of animals[119]. There are three rules that governs each boid in the simulation environment such as cohesion, separation and alignment. There is no external or dynamic change in the environment and the vision of each boid is usually restricted within a range. Cohesion makes each boid to move towards the flock, whereas separation keeps them a distant apart from each other. Each boid only reacts to the movements of its flock mates in the neighbourhood. The concept of neighbourhood is to measure or limit the visibility range of a boid. It is parameterised by a distance and an angle of a boid. The figure 6.7 depicts a sensor range of a boid.

The behaviour of the simulation is that the boids flock together and share common interest rather than individual planning behaviour. It can also move at different speeds. Boids in general is collaborative in nature and move together unless they are hit with an obstacle in the environment. To make it simple, there are no obstacles in the environment. It means that there are no external factors to change or split a group of boids. A snapshot

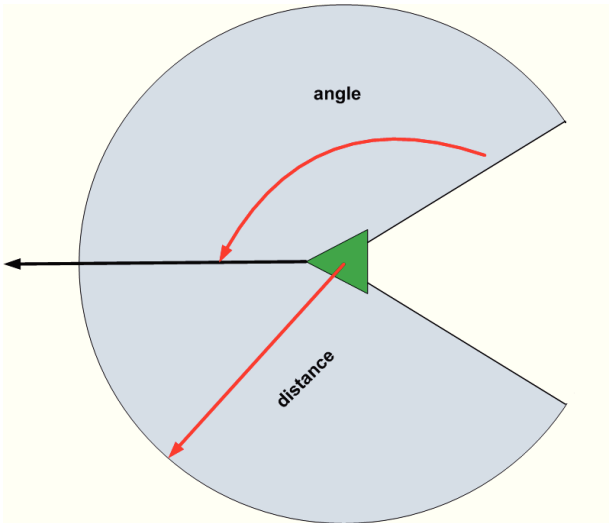


Figure 6.7: Sensor range of a boid.

of Boids simulation is depicted in figure 6.8. The simulation parameters and experimental setup for this simulation is presented in the following section.

6.2.2.1 Experiment Setup

Boids is not highly dynamic since they have co-ordinated motion in the simulation. The tendency of each boid is to find and flock together with neighbouring boids. The specific parameters used for our investigation in the kernel is presented in table 6.7

The experimental setup for this analysis consisted of a CLP-tree of 7 CLPs. Boids themselves are modeled as SSVs. All SSVs were placed at the root CLP to more clearly show the effect State Migration has on performance. 16 ALPs (8 agents/ALP), four attached to each leaf CLP. Simulations are run for 300 logical time ticks (or steps). There are 3 random seeds used for each simulation parameter, so in effect 3.(5.5) = 75 experiments (where each experiment is run for 20 times) were run for this simulation analysis.

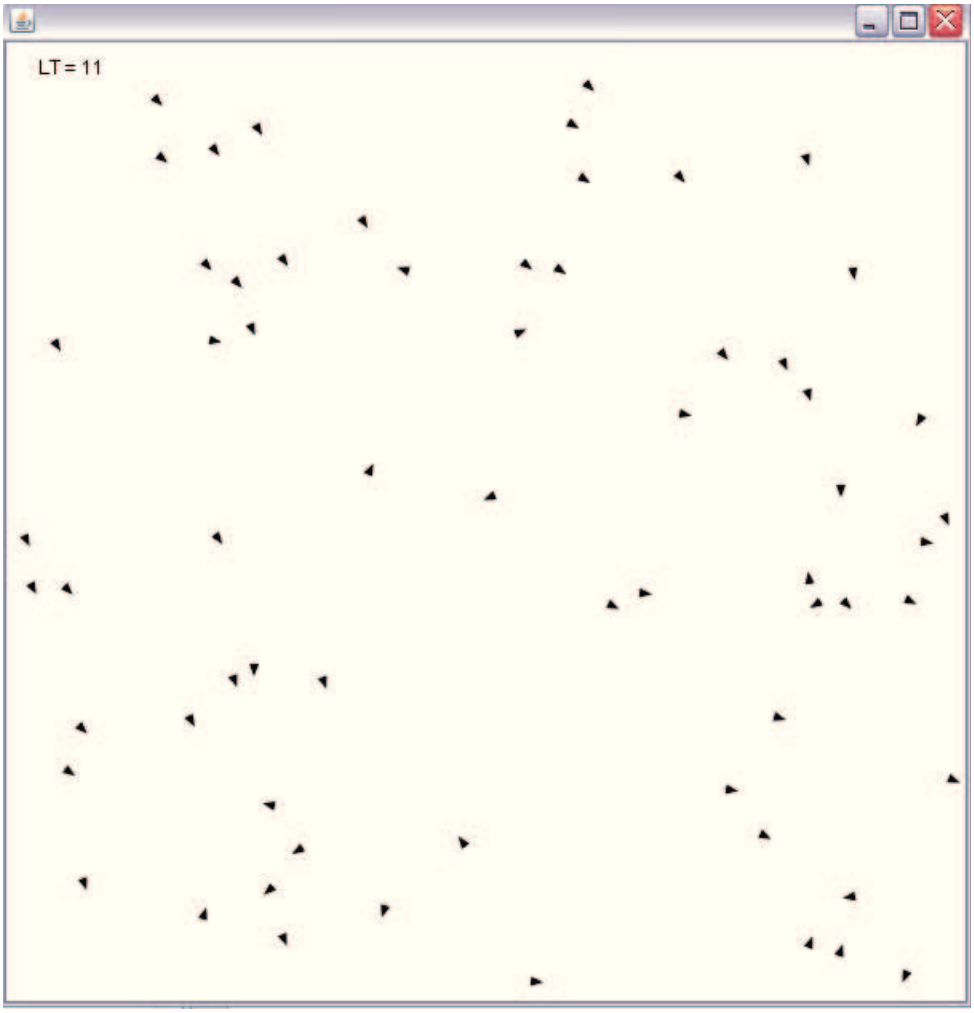


Figure 6.8: A snapshot of Boids simulation.

Table 6.7: Experimental Setup

Environment Size	1000 x 1000
Range Query Size	0.1, 0.2, 0.3, 0.4, 0.5 (Prop. to the environment size)
Velocity	2, 4, 6, 8, 10
Number of Agents	64 (8 agents/ALP)
Separation limit	50
Cohesion weight	2
Alignment Weight	2
Separation weight	5

The investigation focuses on the influence of two simulation parameters such as *RQ size* and *Velocity* in the system measured by three metrics: *Access Cost, Range Query Propagation and Simulation Time*. Results for each of these performance metrics is presented in the following sections.

6.2.2.2 Access Cost and Query/Update propagation

This section presents impact of accessing SSVs under different conditions of experimental setup of Boids simulation. The performance is measured by two metrics: Access Cost and Query/Update propagation. The figure 6.9 presents the total access cost of the simulation for varying *RQ size* and *Velocity* values. As the results were averaged over different experimental runs including 3 different pseudo random number generator seeds and 20 runs of each seed, the standard deviation for each result in the graphs is depicted using error-bars. Without SM, the cost varies from 54683722 (with *RQ size* = 0.1 and *Velocity* = 2) to 20526840042 (with *RQ size* = 0.5 and *Velocity* = 10). With SM, the cost varies from 289646446.67 (with *RQ size* = 0.1 and *Velocity* = 2) to 19047987612.67 (with *RQ size* = 0.5 and *Velocity* = 10).

Without any dilution, the figure clearly shows that cost increases with the increase in *RQ size* and *Velocity* parameter values. The nature of Boids simulation and its experimental parameters have a direct relation with the cost. With smaller values of *RQ size* and *Velocity*, the cost increases linearly. 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. Smaller the range window (sensor range) and slower the movement of agents smaller the flocking groups. It also takes longer time to find neighbours (or flock mates). This means that the number of SSVs within a sensor range increases gradually. But as both parameter values increase, the agents find their neighbours (or flock mates) quite early and never change their neighbourhood throughout the simulation. The formation of groups is fast and more volatile, resulting

to more centralised placement of SSVs in the CLP tree increasing CLP loads and rollback costs. The number of SSVs falling within the range window will be quite large increasing the overlapping access patterns and increases the cost exponentially.

We can also observe that the variance is increasing exponentially with increase in *RQ size* and *Velocity* values. The observance of variance values indicate the volatility of rollbacks and its impact with the cost with varying experimental parameter values and experimental runs. Without SM, the variance of the cost ranges from 95639469.49 (with *RQ size* = 0.1 and *Velocity* = 2) to 5339596723 (with *RQ size* = 0.5 and *Velocity* = 10), whereas the range varies from ranges from 231265610.5 (with *RQ size* = 0.1 and *Velocity* = 2) to 5263974163 (with *RQ size* = 0.5 and *Velocity* = 10). This pattern is further illustrated with the number of SSVs fetched by agents in figure 6.10. Without SM, the number varies from 102709.00 (with *RQ size* = 0.1 and *Velocity* = 2) to 624710.67 (with *RQ size* = 0.5 and *Velocity* = 10). With SM, it varies from 106776.67 (with *RQ size* = 0.1 and *Velocity* = 2) to 632982.00 (with *RQ size* = 0.5 and *Velocity* = 10). The result in the figure 6.10 bears a direct relationship with the result presented in 6.9.

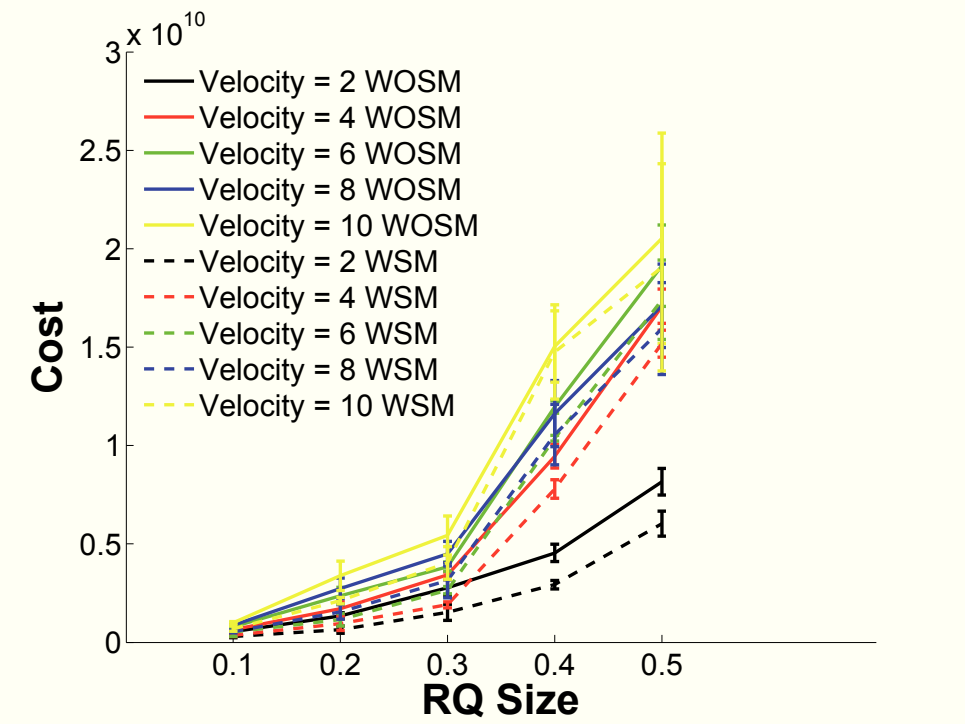


Figure 6.9: Average Cost for varying *RQ size* and *Velocity* values for Boids Simulations with their standard deviations (Error bars).

As expected, the figure also shows that the variance in the cost increases with increasing values of *RQ size* and *Velocity* values. Herewith, the time and frequency of formation of groups with neighbouring boids determines the number of rollbacks on the distributed environment directly impacting the cost and its variance. We also compared the cost with and without SM using the equation 5.2.2 termed as *Cost Reduction*. The table depicted in 6.8 presents *Cost Reduction* in percentage (%) for varying *RQ size* and *Velocity* values of Boids simulation. When *RQ size* and *Velocity* values increases, the cost reduces linearly to a mid-point and then reduces sharply. This is attributed to the fact that the number of SSV accesses increases exponentially as presented in 6.10. It varies from 47.03% (with *RQ size* = 1 and *Velocity* = 2) to 1.87% (with *RQ size* = 1 and *Velocity* = 10).

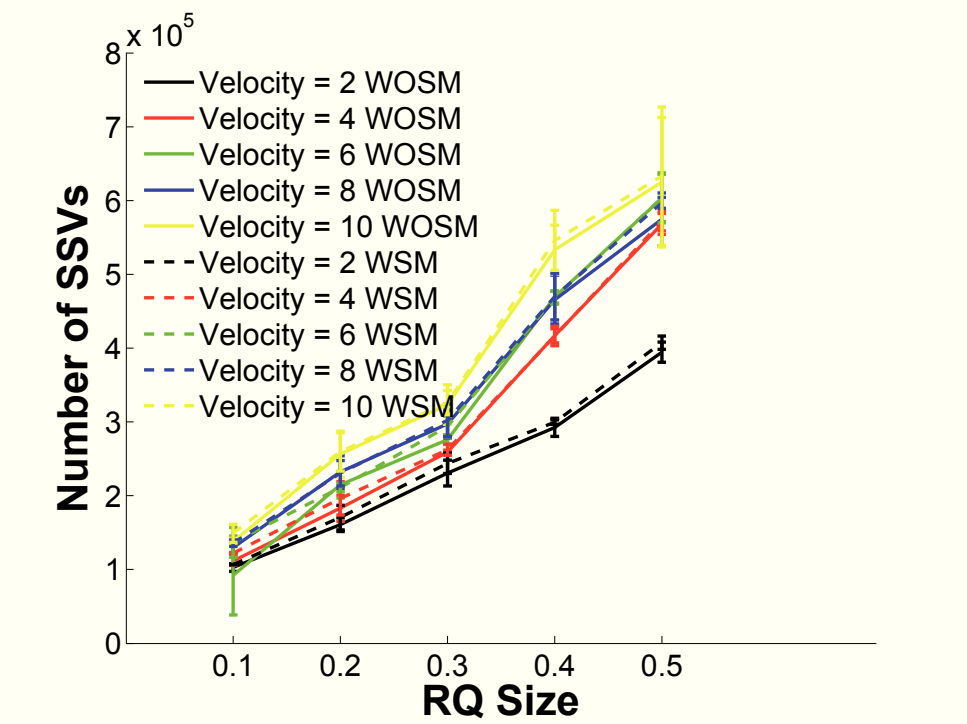


Figure 6.10: Total Number of SSVs fetched for varying *RQ size* and *Velocity* values with their standard deviations.

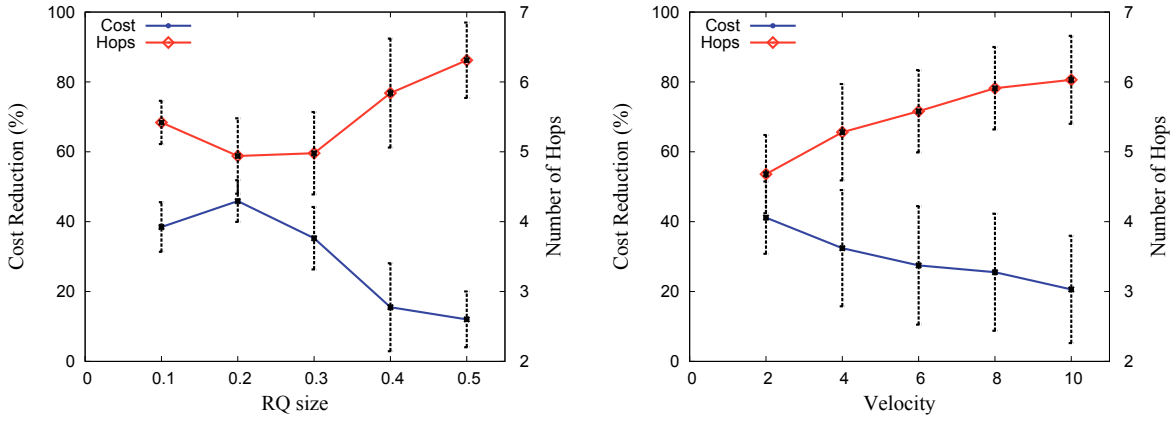
To show more clearly the access patterns of agents in Boids simulation, the figure 6.11a presents an average *Cost Reduction* and number of hops for range queries (With SM) for varying *RQ size* values with their variance over values of *Velocity* and figure 6.11b presents an average *Cost Reduction* and number of hops for range queries (With SM) for varying

Velocity values with their variance over values of *RQ size*. Since SSVs are placed at root CLP, the number of hops Without SM would be constant 7. With SM, the expectation is that SSVs move towards the accessing agents thereby reducing the hops.

As *RQ size* and *Velocity* values are increased, the number of hops required to access SSVs increases linearly thereby reducing *Cost Reduction* (or increases access cost With SM). This is evident from figure 6.11b as query propagation increases from 4.68 to 6.03 with its deviation and cost reducing from 41.15% to 20.6%. This pattern is also observed in figure 6.11a. This is because the number of SSVs within the neighbourhood of boids increases linearly with overlapping access patterns. Overall reduction in query propagation for different parameter values is 5.5. The number of update (write) hops With SM for all values of *RQ size* and *Velocity* of Boids simulation is just 1. Agents in this simulation only updates themselves (all the time) as the agent does not move any other objects or agents. The overall gain in cost on average for Boids simulation is almost 30%.

RQ size	***Velocity***				
	2	4	6	8	10
0.1	47.03	44.25	32.42	37.8	30.85
0.2	51.88	44.93	51.42	43.81	37.47
0.3	45.37	44.16	31.18	29.99	25.58
0.4	35.49	17.6	13.28	9.23	1.87
0.5	25.99	11.13	9.07	6.74	7.2

Table 6.8: *Cost Reduction* in percentage for varying *RQ size* and *Velocity* values of Boids Simulation.



(a) . (b) .

Figure 6.11: Average *Cost Reduction* and number of range query hops for varying *RQ size* and *Velocity* with their standard deviations.

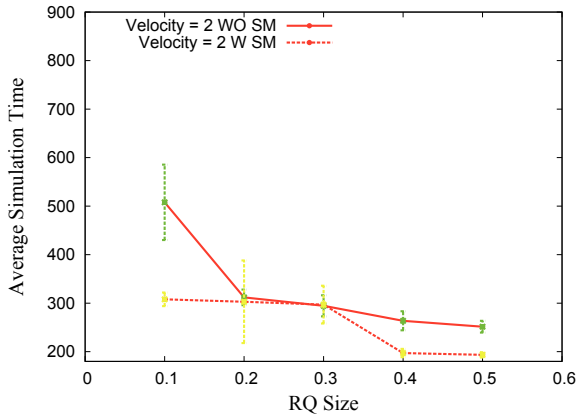
The results presented so far measures the access cost and query/update propagation for Boids simulation. It shows that overall reduction in cost for varying different values of *RQ size* and *Velocity* is 30%. We also have shown that range query propagation increases with increasing *RQ size* and *Velocity* values. With SM, the overall reduction on average in query propagation is 5.5 and update propagation is 1 (as compared Without SM is 7). So, the performance of the system, in terms of cost and propagation, has improved substantially.

6.2.2.3 Simulation Time

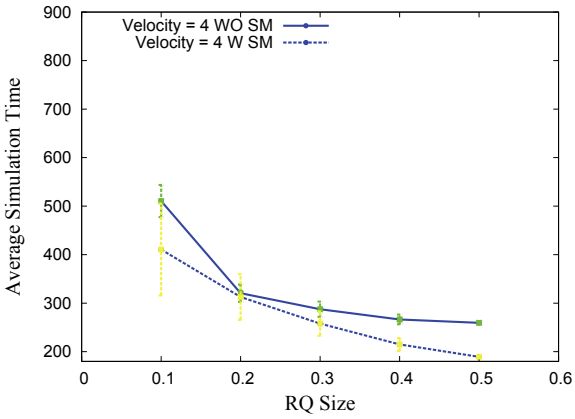
This section presents the impact of state migrations under different conditions of experimental setup of Boids simulation on simulation time. The figure 6.12 presents an average simulation time for varying *RQ size* and *Velocity* values of boids simulation. As the results were averaged over different experimental runs including 3 different pseudo random number generator seeds and 20 runs of each seed), the standard deviation for each result in the graphs is depicted using error-bars. Without SM, the time varies from 510.48 (with *RQ size* = 0.1 and *Velocity* = 4) to 238.22 (with *RQ size* = 0.5 and *Velocity* = 10). With SM, the time varies from 545.25 (with *RQ size* = 0.1 and *Velocity* = 6) to 189.3 (with *RQ*

size = 0.5 and *Velocity* = 4). The highest gain or reduction in time With SM is 200.01 (with *RQ size* = 0.1 and *Velocity* = 2).

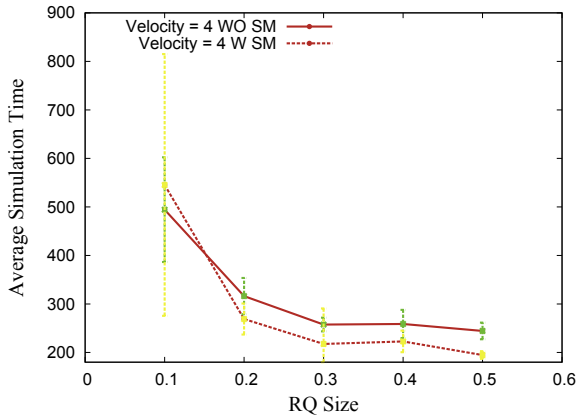
The figure shows that the simulation time decreases linearly with increasing *RQ size* and *Creation Probability* values with their standard deviations. A simulation progress depends on the progress of agents in time (both real and logical). Agents progress can be hindered with the rollbacks. We quantified this effect with the calculation of an average *rollback depth* incurred by agents. A *rollback depth* is the difference in logical time between rollback time and agent's LVT (Local Virtual Time). Higher the depth of rollback, longer the difference in progress between agents and longer it takes to finish experiments. With minimum *RQ size*, the formation of flocking groups in the simulation is slow as boids find their new neighbours. In this case, agents progressing at their own rate are hindered with higher rollback depth as they form groups slowly (with their access patterns overlap). On the other hand, as *RQ size* increases with *Velocity*, the groups are formed quite early and they progress in similar pace (in logical time) with overlapping access patterns.



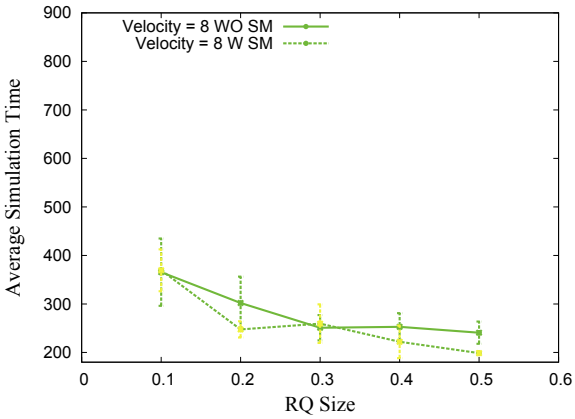
(a) *Velocity* = 2



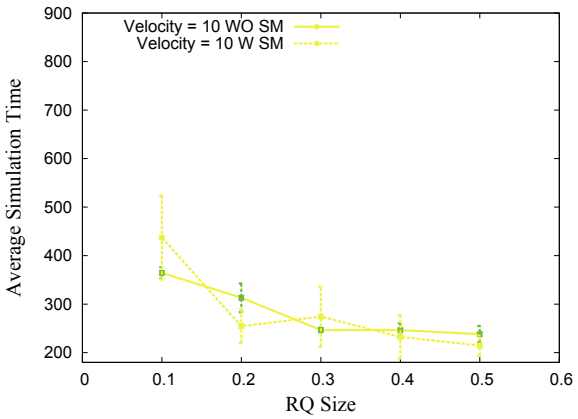
(b) *Velocity* = 4



(c) *Velocity* = 6

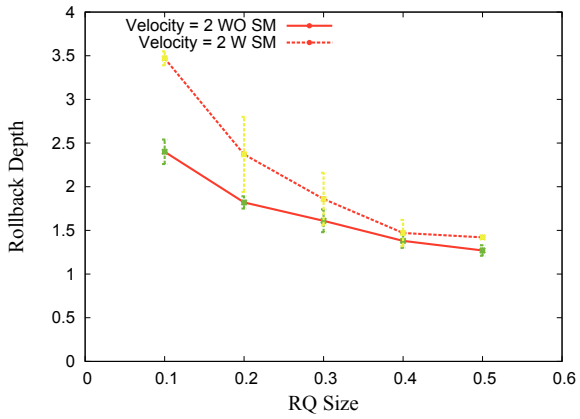


(d) *Velocity* = 8

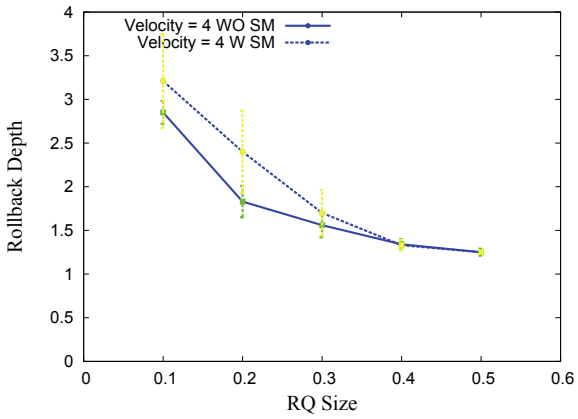


(e) *Velocity* = 10

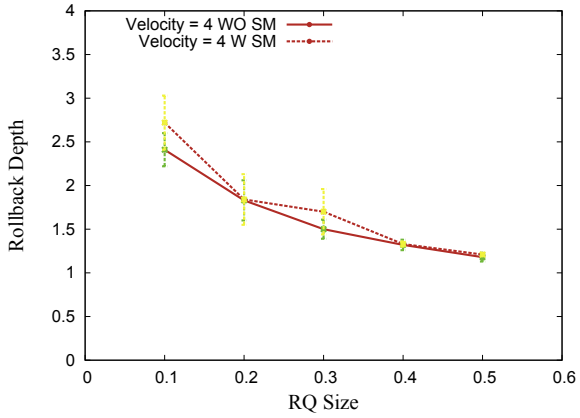
Figure 6.12: Average Simulation Time for varying *RQ size* values with their standard deviations (Error bars).



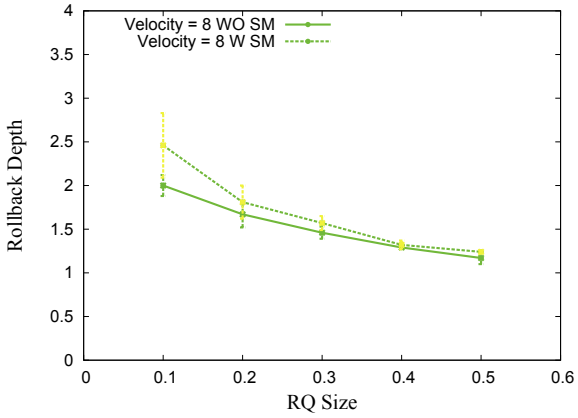
(a) *Velocity* = 2



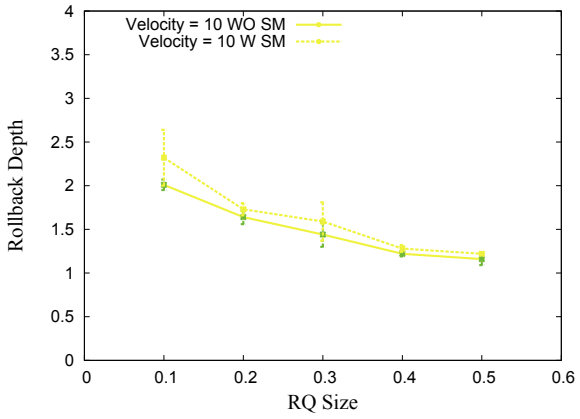
(b) *Velocity* = 4



(c) *Velocity* = 6



(d) *Velocity* = 8



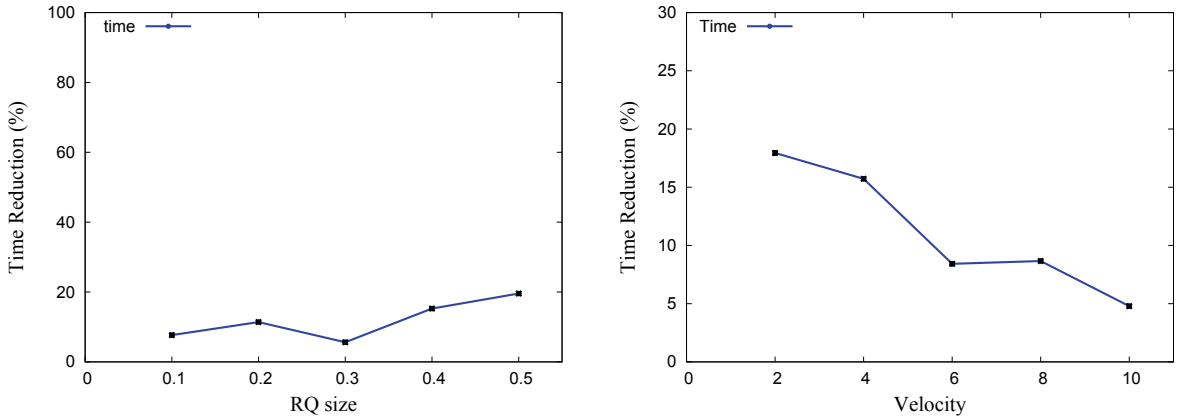
(e) *Velocity* = 10

Figure 6.13: Average Rollback Depth for varying *RQ size* values with their standard deviations (Error bars).

This behaviour affects the simulation progress of agents in LVT, as agents with smaller

range window progress independently and any overlap in access (as they form groups quite late) will result in a deeper rollback. But as *RQ size* increases, more cohesive progress in LVT is observed with a minimal rollback depth. This is observed in the figure 6.13 which presents an average rollback depth of an agent for varying *RQ size* and *Velocity* values of boids simulations with their deviations. This bears out the expectation that the average depth decreases with increasing *RQ size* and *Velocity* values. Without SM, it varies from 2.8 (*RQ size* = 0.1 and *Velocity* 4) to 1.1 (*RQ size* = 0.5 and *Velocity* = 10) and With SM, it varies from 3.8 (*RQ size* = 0.1 and *Velocity* 4) to 1.2 (*RQ size* = 0.5 and *Velocity* = 10).

The extra rollbacks incurred With SM is compensated with reduction in query and updates propagation as presented in section 6.2.2.2, thereby contributing to the reduction in simulation time (as presented in figure 6.12). We also have compared the times with and without SM using a ratio *Time Reduction* (similar to equation 5.2.2). Figure 6.14 presents an average *Time Reduction* for varying values of *RQ size* and *Velocity* with their standard deviations.



(a) . (b) .

Figure 6.14: Average *Time Reduction* for varying *RQ size* and *Velocity* with their standard deviations.

The figure clearly shows that as *RQ size* increases, *Time Reduction* increases from 7.68 to 19.56 and its variance decreases from 0.21 to 0.06 as rollback depth and query propagation decreases with increasing *RQ size* and *Velocity* values. And predictably when

Velocity increases, *Time Reduction* decreases from 17.94 to 4.79 with its deviation.

6.2.2.4 Summary

The results presented so far in sections 6.2.2.2 and 6.2.2.3 attribute to the fact that the performance of the system, in terms of cost, propagation and time, can be reduced With SM. Boids are mostly cohesive and by nature flocking mates (group) formation is the ultimate goal in the simulation. There are no external factors that influence the agent behaviour. The cost and query/update propagation presented in section 6.2.2 are quite predictable with different parameter values. The cost on average With SM has 30% reduction, query propagation is reduced to 5.5 and update is reduced to just 1. The behaviour of agents in boids simulation played a vital role in determining the rollback patterns. It shows that simulation progress is hindered when the rollback depth is quite high. Though the depth is even higher With SM, the effect is diminished with the reduction in query and update propagation. Overall gain in simulation time on average is 11%.

6.3 Scalability Analysis

Having analysed the interplay of different workload parameters on the performance of the system in section 3.4, this section focuses on the scalability of PDES-MAS for varying Number of ALPs and Number of CLPs (depth of the PDES-MAS tree). As Number of ALPs increases, the volume of data access in the CLP tree increases too. Following the philosophy of PDES-MAS, the expectation is that for a certain Number of ALPs distributing the SSVs across the CLP tree would increase the performance of the system since migrating state reduces access costs and rollbacks. The extra time incurred by rollbacks (due to range updates and state migrations) will be compensated with localised data access. However, as Number of CLPs increases even further we expect a cut off point after which communication takes over, overheads increase and performance degrades.

The experiments have been carried out in BlueBEAR cluster environment. 20 worker

nodes were used for experimentation with each node has dual-processor dual-core (4 cores/node) 64-bit 2.6 GHz AMD Opteron giving 8GB of memory. All worker nodes are running Scientific Linux 5.2. *openMPI (1.4.3)*[40] libraries for experiments to launch and establishing communication between processes. An overview of TileWorld simulation is already presented in section 6.2.1. The experiment setup of PDES-MAS kernel and simulation parameters of TileWorld used for this investigation is presented in the following section.

6.3.1 Experiment setup

This section presents the experiment setup and simulation parameters of TileWorld simulation for scalability analysis. The table 6.9 presents the simulation parameters of TileWorld simulation under investigation.

Table 6.9: Simulation Parameters

Environment Size	50 x 50
Range Query Size	2
Creation Probability	.5
Object life time	50
Number of Agents	16, 32, 64, 96, 128, 256

The experimental setup for this analysis consists of a CLP-tree with varying depths. The objects (Tiles, holes, rocks and agents themselves) are modeled as SSVs. All SSVs were placed at the root CLP to more clearly show the effect distribution has on performance. ALPs are attached as leaves and is varied for each experiment. Simulations are run for 300 logical time ticks (or steps). There are 3 random seeds used for each simulation parameter, so in effect 3.(5.6) = 90 experiments (where each experiment is run for 20 times) were run for this simulation analysis. The table 6.10 presents the experimental setup for this investigation.

Table 6.10: Experimental Setup

Number of CLPs	1, 3, 7, 15, 31
Number of ALPs	16, 32, 64, 96, 128, 256

The evaluation focuses on the scalability of the framework with varying Number of CLPs and Number of ALPs in the simulation. As we increase Number of ALPs, it increases volume of data access in the CLP tree. The expectation is that varying Number of CLPs with distributing SSVs around the CLP tree would increase the performance of the system. We also expect that migrating state reduces access times and rollbacks thereby reducing the simulation time. The extra time incurred by rollbacks (due to range updates and state migrations) will be compensated with localised data. However, as we increase Number of CLPs and Number of ALPs, we expect to see a performance degrade as the overhead increases with the migration itself.

6.3.2 Results

This section presents an analysis on the performance of PDES-MAS framework for varying Number of ALPs and Number of CLPs using TileWorld simulation. The performance is measured over different metrics: *Simulation time, Range Query Response time*, *Query and Update hops* and *Rollbacks*. Comparisons are made, where required, to show effects of dynamic distribution (i.e., With SM) on the performance of the system. The simulation time is the total elapsed time of an experimental run in the simulation. The figure 6.15 presents the average simulation time for varying Number of ALPs and Number of CLPs with their standard deviations.

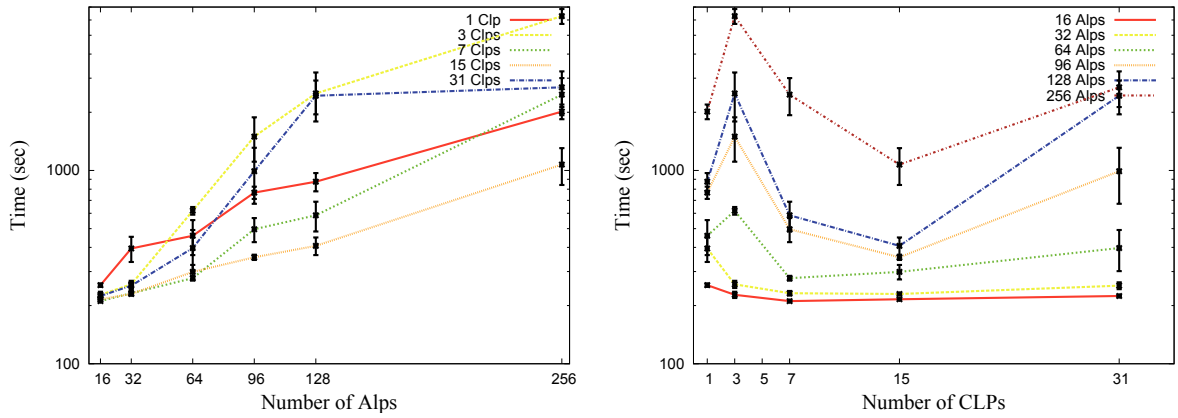


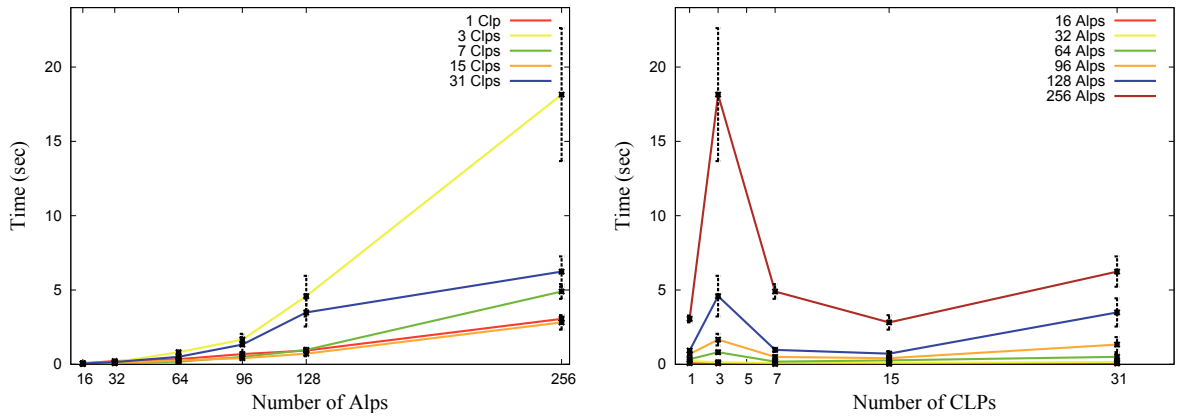
Figure 6.15: Average Simulation Time for varying Number of ALPs and Number of CLPs with their standard deviations.

Figure 6.15a shows that simulation time increases with increasing Number of ALPs in the tree. With Number of CLPs = 1 (where dynamic distribution is disabled), the simulation time increases from 254.95 (with Number of ALPs = 16) to 2015.4 (with Number of ALPs = 256). With 3 CLPs, simulation time is much higher as the distribution of the SSV load is not enough to amortise the communication and management overhead introduced by SM. The same phenomenon is observed for 7 CLPs and 31 CLPs (much sooner) where SSVs become too thinly distributed across the CLP-tree making the system communication bound. A tree of 15 CLPs is obviously the optimal configuration for this particular workload set up.

Similar patterns are observed in figure 6.15b. For a small number of ALPs, moving from 1 CLP to 7 CLPs reduces simulation time which then remains stable - suggesting that SSV accesses are totally localised. For higher number of ALPs, going from 1 to 3 CLPs increases simulation time as the SSV distribution is not enough to amortise the communication and management overheads introduced. After that, we observe a speedup for increasing tree depth up to a point after which communication overhead takes over increasing the simulation time. Clearly, the higher the number of ALPs, the longer it takes before the system becomes communication bound.

With Number of CLPs = (3,7,15,31) (where dynamic distribution enabled), the sim-

ulation time increases from 210.78 (with Number of CLPs = 7 and Number of ALPs = 16) to 6280.38 (with Number of CLPs = 3 and Number of ALPs = 256). This is because increasing Number of ALPs linearly increases the number of accesses issued in parallel to the CLP tree. This is clearly evident for all depths of Number of CLPs. Since PDES-MAS architecture extracts information from the CLP tree using queries, it increases the overhead of handling queries arriving in a CLP at different ports at the same time. This is evident from the calculation of *Range Query Response times*. A *Range Query Response time* is the time taken for a range query issued from an ALP to parse the CLP tree and gets back with a result. Figure 6.16 presents an average *Range Query Response times* for varying Number of ALPs and Number of CLPs values with their standard deviations.



(a) (b)

Figure 6.16: Average Range Query Response time for varying Number of ALPs and Number of CLPs with their standard deviations.

The response times presented in figure 6.16a is clearly inline with the figure presented in 6.15a. It shows that the query response times increase linearly with increasing Number of ALPs. So far we have observed a predictable pattern of varying Number of ALPs in the investigation. However, figure 6.15b clearly shows the effect of varying both Number of CLPs and Number of ALPs experimental values.

We can observe three different scenarios. First, it shows that when Number of ALPs are minimum, increasing depth of Number of CLPs decrease the simulation time linearly

as the state migration algorithm moves SSVs closer to the accessing ALPs and also distributing SSVs across the tree for parallel access. This is evident on observing simulation times when Number of ALPs varies from 16 to 64 where dynamic distribution reduces time for varying depths of Number of CLPs in comparison with Number of CLPs = 1. Second, when Number of CLPs are minimum, increasing Number of ALPs increases the simulation time exponentially because of the overhead created with state migration (or poor distribution). This pattern is observed for simulation times when Number of ALPs values varying from 64 to 256 for Number of CLPs = 3. And final observation is that as we increase both Number of CLPs and Number of ALPs, we can observe the simulation time decreasing linearly to an optimum value and then increases again. This is because as the depth of CLP tree increases, state migration takes much longer time to move SSVs closer to the ALPs. This is observed with Number of CLPs = $(7, 15, 31)$ for varying Number of ALPs. A distribution is optimum if it reduces the overhead of handling messages and localising data access (or access times) thereby reducing the simulation time.

To show the effect of dynamic distribution (With SM) on data (SSVs) access, we present a number of hops required for an agent to access (query/update) a SSV/SSVs. For this evaluation, we have placed SSVs at top (root CLP) of the tree. So, With SM, an average number of hops to reach root CLP for different depths of CLP tree is calculated as depth plus one and their values would be $(1, 3, 5, 7, 9)$. With dynamic distribution (SMs), SSVs are distributed around the CLP tree, the hops could vary between $(1 < h < (2 * d_t))$, where h is number of hops and d_t is depth of a CLP tree. But the expectation is that SSVs are moved towards the accessing agents localising their access and decreasing the propagation of a query/update.

Tables 6.11 and 6.12 present an average number of hops for a range query and an update for varying Number of CLPs and Number of ALPs with their standard deviations (in brackets). The benchmark used for our evaluation is TileWorld, which is highly dynamic and explores new grid cells every step. So, the number of hops for range queries is average whereas updates are mostly localised for varying Number of ALPs and Number

Number of ALPs	Number of CLPs				
	1	3	7	15	31
16	1	2.33	4.33	6.33	9
	(0)	(0.48)	(0.48)	(0.48)	(0)
32	1	2.33	4.33	6.33	9.33
	(0)	(0.48)	(0.48)	(0.48)	(0.48)
64	1	2	4	6	10
	(0)	(0)	(0)	(0)	(0)
96	1	2	4.33	6	10.33
	(0)	(0)	(0.48)	(0)	(0.48)
128	1	2.33	5	6.67	10.33
	(0)	(0.48)	(0)	(0.48)	(0.48)
256	1	2.33	5	6.67	10.33
	(0)	(0.48)	(0)	(0.48)	(0.48)

Table 6.11: Number of Range Query Hops for varying Number of ALPs and Number of CLPs values with their standard deviations within brackets.

Number of ALPs	Number of CLPs				
	1	3	7	15	31
16	1	1.33	3	4	5.33
	(0)	(0.48)	(0)	(0)	(0.48)
32	1	1	2	2.67	3.67
	(0)	(0)	(0)	(0.48)	(0.48)
64	1	1	1	1	2
	(0)	(0)	(0)	(0)	(0)
96	1	1	1	1	2
	(0)	(0)	(0)	(0)	(0)
128	1	1	1	1	2
	(0)	(0)	(0)	(0)	(0)
256	1	1	1	1	2.33
	(0)	(0)	(0)	(0)	(0.48)

Table 6.12: Number of Update Hops for varying Number of ALPs and Number of CLPs values with their standard deviations within brackets.

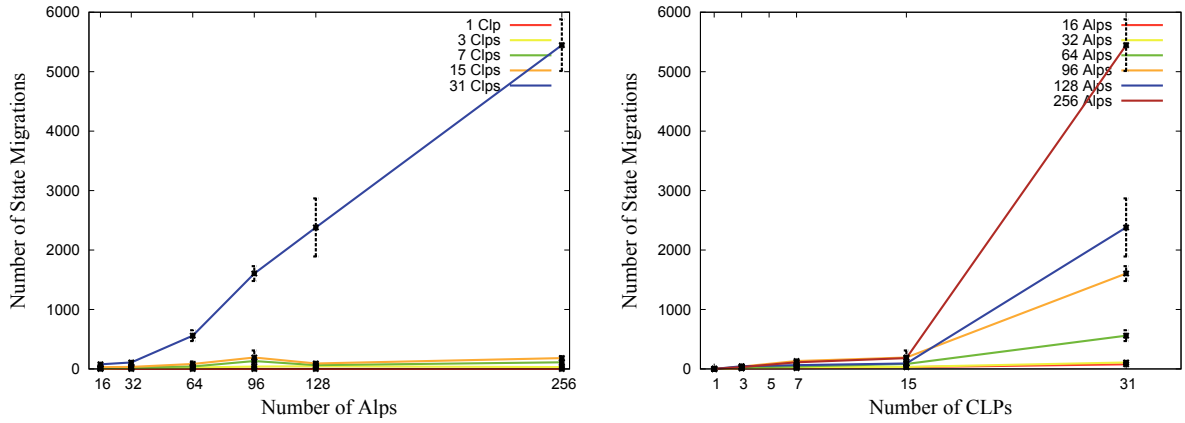
of CLPs.

However, what remains to be seen is that whether initiating state migration algorithm has increased or reduced the communication overhead of the system. To quantify the effect, we have populated several types of messages exchanged between ALPs and CLPs such as queries and update (to extract and update information), rollbacks (to correct ordering of events), State Migration (to distribute SSVs) and GVT (to garbage collect

a portion of shared state). Of these messages, rollbacks bear a direct relation to the progress of the simulation in time (both logical and real). This is because as the number of rollbacks increases, it increases the number of accesses (queries/updates) regenerated thereby increasing the simulation time of an experiment.

Typically, a CLP triggers a rollback when a *straggler write* arrives on a SSV invalidating premature reads or triggers indirectly through *straggler range updates*. In addition to that, rollbacks are also triggered on initiating State Migration algorithm. To quantify this effect, figures 6.17 and 6.18 present the number of State Migrations (SM) initiated and number of Range Updates (RUs) generated in the simulation and figure 6.19 presents the number of rollbacks generated by CLPs for varying Number of ALPs and Number of CLPs with their standard deviations.

The number of SMs initiated increases linearly with increasing Number of CLPs and Number of ALPs in the simulation. Higher the depth of a CLP tree, larger the number of SMs required to migrate SSVs across the CLP tree. Predictably, the number of SMs initiated varies from 8 (with Number of CLPs = 3 and Number of ALPs = 16) to 5445 (with Number of CLPs = 31 and Number of ALPs = 256). It also shows that as Number of CLPs increases, there is an exponential increase in the number of state migrations.

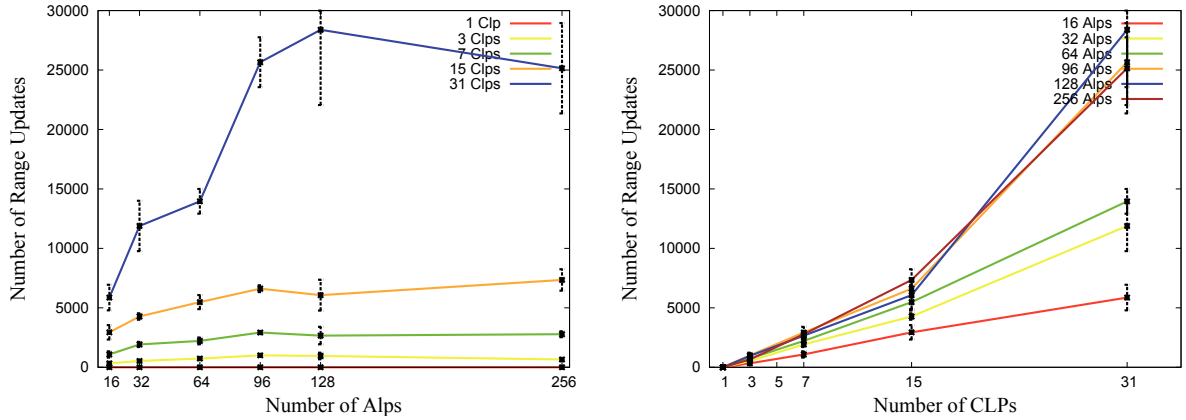


(a) (b)

Figure 6.17: Number of State Migrations (SM) initiated for varying Number of ALPs and Number of CLPs with their standard deviations.

The same pattern can be observed for range updates generated by CLPs. Range

Updates are generated when there is a change in the value of a SSV in a CLP (either due to updates or State Migration moving SSVs) has an influence on the external ports. It means the range window of a CLP has either shrunken or expanded (explained in detail in chapter 4). Figure 6.18 presents the number of Range Updates generated for varying Number of CLPs and Number of ALPs with their standard deviations. It varies from 331 (with Number of CLPs = 3 and Number of ALPs = 16) to 25150 (with Number of CLPs = 31 and Number of ALPs = 256). It bears out the expectation that as the depth of CLP tree increases the number of range updates generated increases linearly.



(a) (b)

Figure 6.18: Number of Range Updates (RU) generated for varying Number of ALPs and Number of CLPs with their standard deviations.

With the observance of initiated State Migrations in figure 6.17 and number of range updates generated in figure 6.18, we would expect the possibility of rollbacks generated from both scenarios (including straggler writes) to follow the same pattern. Figure 6.19 presents the number of rollbacks generated for varying Number of ALPs and Number of CLPs with their standard deviations. The number of rollbacks presented here is the sum of the rollbacks generated through all scenarios mentioned above. When Number of CLPs = 1, neither range updates nor SMs are initiated. The number of rollbacks increases linearly with increasing Number of ALPs with high volume of data access issued in parallel. When Number of CLPs = 1, it varies from 545.3 (with Number of ALPs = 16) to 24846.3 (with Number of ALPs = 256). Whereas with dynamic distribution enabled,

rollbacks generated varies from 231.6 (with Number of CLPs = 7, Number of ALPs = 16) to 43486 (with Number of CLPs = 31, Number of ALPs = 256). We can observe that when Number of ALPs are minimum, the number of rollbacks generated with all scenarios (mentioned above) are fewer in comparison with rollbacks generated in case of Number of CLPs = 1. This shows that distribution improves agents progression in simulation time (logical) thereby reducing possibilities of occurrence of straggler updates (writes). We can also observe that minimum or optimum number of state migrations result in reduction of rollbacks for varying data access (as observed in when Number of CLPs = 15).

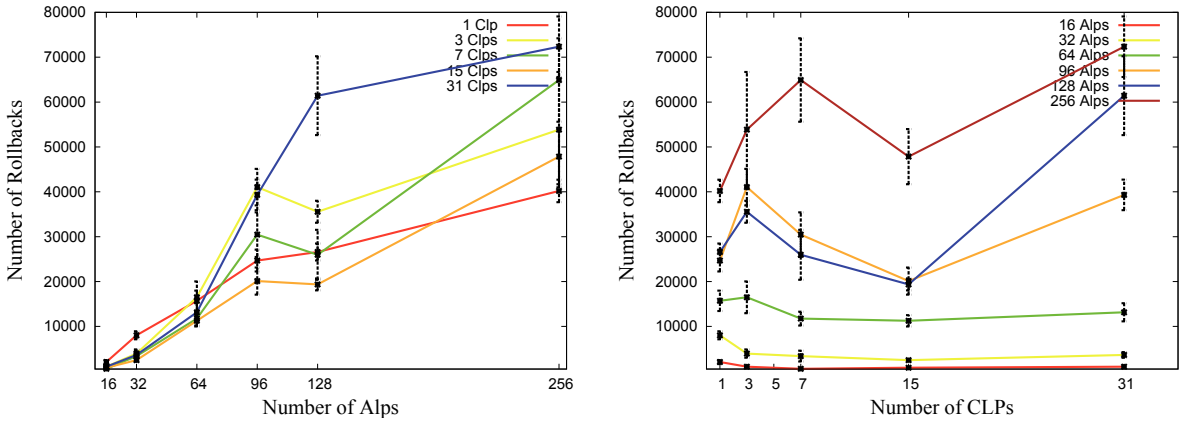


Figure 6.19: Number of rollbacks generated for varying Number of ALPs and Number of CLPs with their standard deviations.

But not all rollbacks could be valid, for example, a rollback arriving at time $t´$ is neglected if the agent is already been rolled back to logical time $< t´$. So, the valid rollbacks in an ALP are measured as rollbacks committed. Figures 6.20 presents the number of rollbacks committed for varying Number of CLPs and Number of ALPs with their standard deviations.

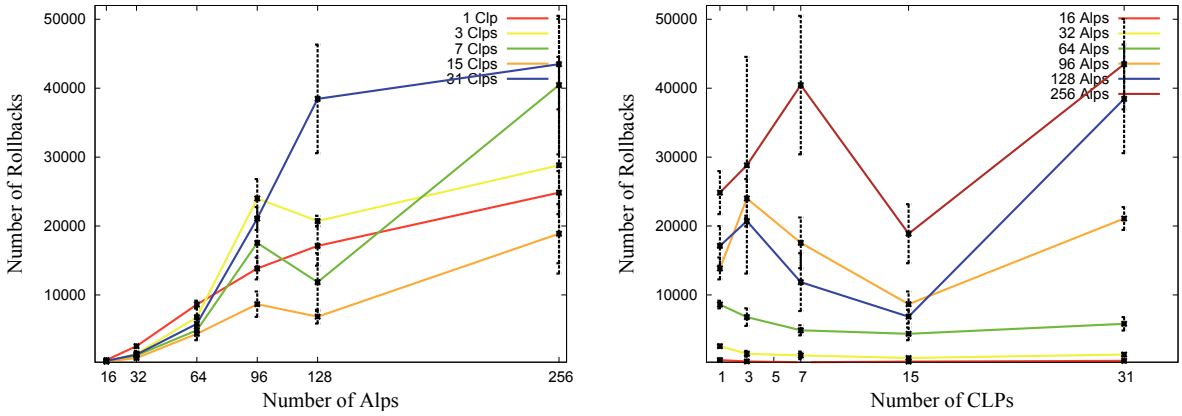


Figure 6.20: Number of rollbacks committed for varying Number of ALPs and Number of CLPs with their standard deviations.

It shows that the number of rollbacks committed increases with increasing Number of CLPs and Number of ALPs with their standard deviations. It varies from 231.7 (with Number of CLPs = 7 and Number of ALPs = 16) to 43486 (with Number of CLPs = 31 and Number of ALPs = 256). The pattern has a direct relation to the simulation time presented in figure 6.15 and bears out the expectation that the number of rollbacks committed is fewer than generated rollbacks presented in figure 6.19. To quantify this effect, we compared the rollbacks (generated by CLPs and committed by ALPs) termed as *Acceptance Rate*(A_r), which is the ratio of number of rollbacks committed N_{rc} to the number of rollbacks generated N_{rg} and calculated as given below.

$$A_r = \frac{N_{rc}}{N_{rg}} \cdot 100 \tag{6.1}$$

The ratio gives a measurement of number of valid rollbacks in the simulation. Figure 6.21 presents the acceptance rate (A_r) of rollbacks for varying Number of CLPs and Number of ALPs averaged over different Number of ALPs and Number of CLPs with their variances shown in error bars. Figure 6.21a shows that rate increases as we increase Number of ALPs with their standard deviations. Figure 6.21b shows that as we increase Number of CLPs, rate decreases linearly with their standard deviations to an optimum

(with Number of CLPs = 15) and then increases again. The pattern is inline with the rollbacks and also shows that lower acceptance rate with its deviation contributes to lower simulation time (presented in figure 6.15).

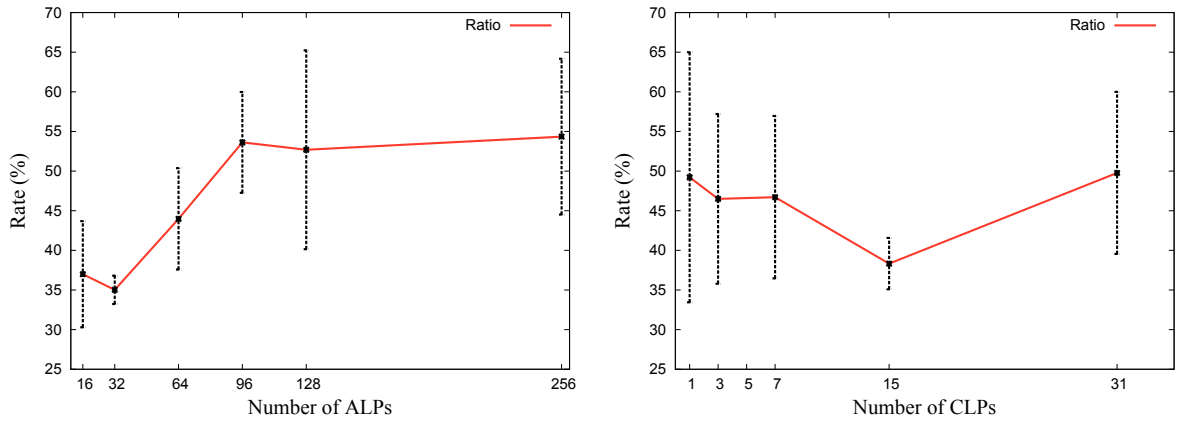


Figure 6.21: Acceptance rate for varying Number of ALPs and Number of CLPs with their standard deviations.

A committed rollback at time t regenerates all events with time $> t$. As the number of rollbacks committed increases, the possibility of regenerating events increases. Figure 6.22 presents a number of Queries/Updates generated (including regenerated due to rollbacks) for varying Number of CLPs and Number of ALPs with their standard deviations. It shows that the number of Queries/Updates generated increases linearly with the increase in rollbacks presented in figure 6.20. It varies from 7696 (with Number of CLPs = 7 and Number of ALPs = 16) to 178791 (with Number of CLPs = 31 and Number of ALPs = 256).

To provide an indication of the overhead incurred in the system in terms of communication, figure 6.23 presents the total number of messages generated or exchanged between ALPs and CLPs (i.e. the bandwidth consumption) for varying Number of CLPs and Number of ALPs in the system. Comparing the bandwidth when Number of CLPs = 1 with varying Number of CLPs, it can be seen that that migrating state in fact reduces the number of rollbacks generated with straggler updates and total number of queries/update messages generated (observed in Number of ALPs = (16 to 96)). The optimum setup is,

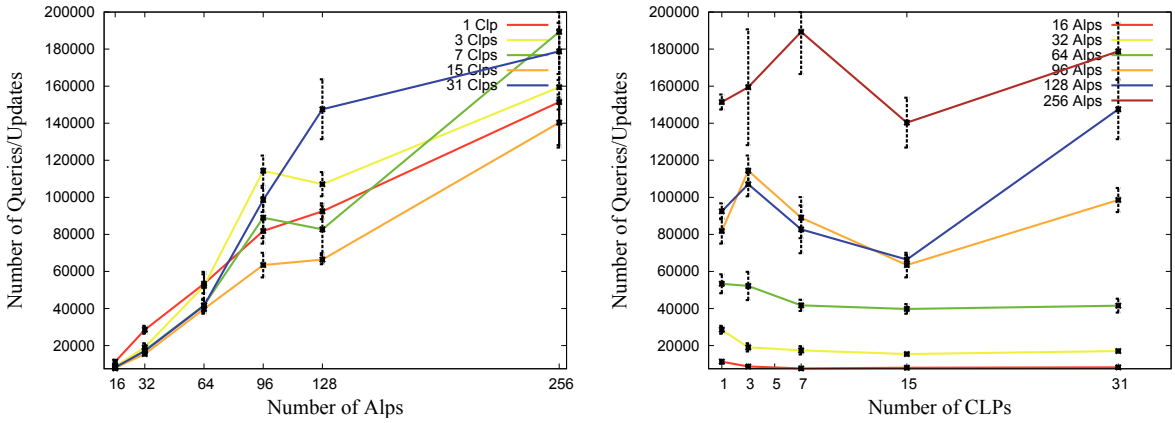
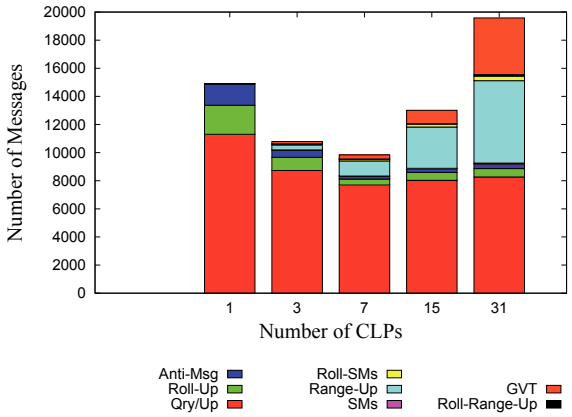
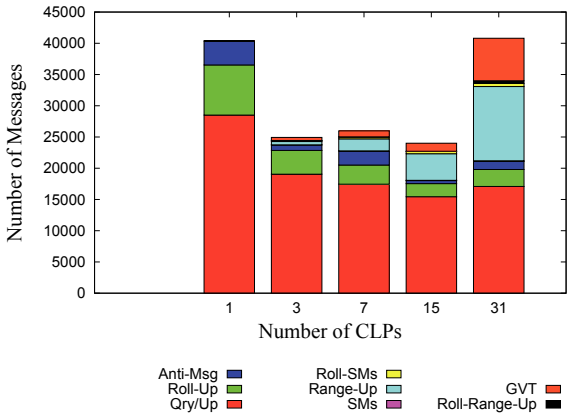


Figure 6.22: Number of queries/updates generated for varying Number of ALPs and Number of CLPs with their standard deviations.

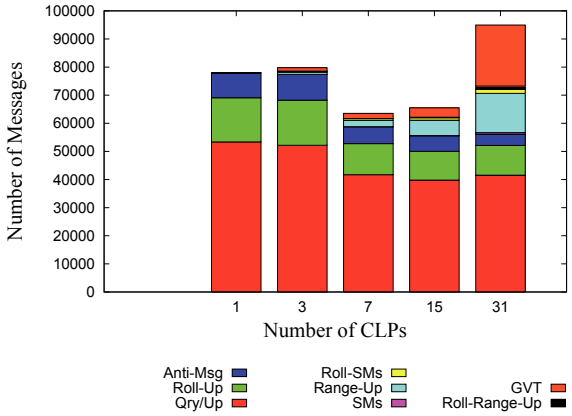
in this case, when Number of CLPs = 15, with the number of queries/updates reduced by 26% and committed rollbacks by 24% for varying Number of ALPs.



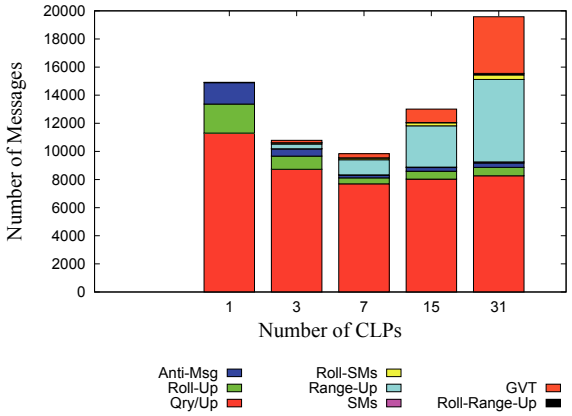
(a) Number of ALPs = 16



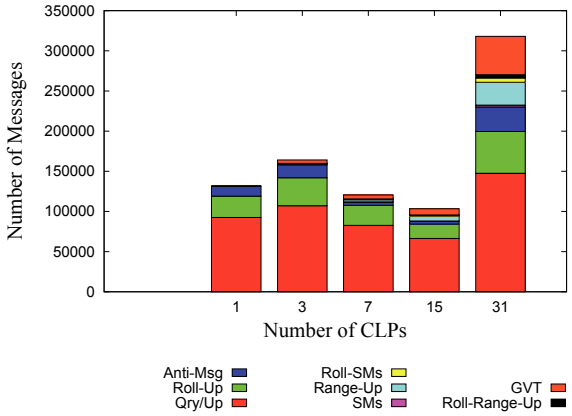
(b) Number of ALPs = 32



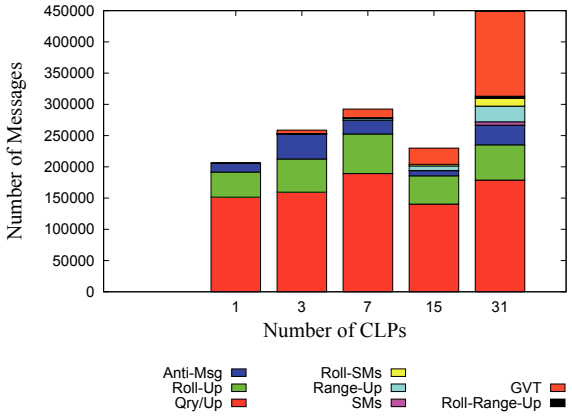
(c) Number of ALPs = 64



(d) Number of ALPs = 96



(e) Number of ALPs = 128



(f) Number of ALPs = 256

Figure 6.23: Total Number of Messages generated for varying Number of ALPs and Number of CLPs with their standard deviations. Qry/Up represents Queries/updates, Roll-Up represents Rollbacks by Updates, Anti-Msg represents Anti-Messages, SMs represents State Migrations, Range-Up represents Range Updates, Roll-SMs represents Rollbacks by State Migrations, Roll-Range-Up represents Range Updates and GVT represents GVT messages.

In general, comparing the bandwidth when Number of CLPs = 1 with varying Number of CLPs, it shows that migrating state in fact reduces the number of rollbacks generated with straggler updates and total number of queries/update messages generated (observed in Number of ALPs = (16 to 96)). The optimum setup is, in this case, when Number of CLPs = 15, the number of queries/updates is reduced by 26% and committed rollbacks by 24% for varying Number of ALPs.

6.3.3 Summary

So far in section 6.3 we have presented an evaluation on the scalability of PDES-MAS framework using TileWorld simulation. We have compared the performance varying Number of CLPs with 1 CLP tree configuration to show the effect of dynamic distribution in the system. The analysis showed that the data access can be localised for varying Number of CLPs and Number of ALPs in the simulation. Also it can be observed that the rollbacks triggered by straggler writes can be reduced with migrating a portion of shared state closer to the agents accessing them. We can also see that as the depth of CLP tree increases to the maximum, the overhead of migration degrades the performance as it increases bandwidth and takes longer time taken to migrate SSVs closer to the agents. This trend might vary if we run the simulation for much longer (logical time steps) or with even larger configurations. Unfortunately, we do not have the resource to test with those configurations. However, it provides pointers with a benchmark analysis and an insight into the performance that distributing SSVs across the tree does improve the simulation time without affecting the performance of the system.

CHAPTER 7

MWGRID: USE-CASE

This chapter presents a working implementation of a project called MWGrid[1] (Medieval Warfare on the Grid) that seeks to address the problems of military logistics of the Battle of Manzikert in 1071 AD using agent based modelling and distributed simulations. PDES-MAS provides a distributed simulation architecture and a Middleware framework acting as a glue between PDES-MAS and the MAS simulation (MWGrid). Basically, the idea of Middleware framework is to provide a light-weight solution to translate events to and from MAS simulation into the format of PDES-MAS framework. I have been involved with this project and contributed towards the development and analysis of the system. Here, a summary of design details of Middleware framework and MWGrid project is presented in the following sections (contents taken and adapted from publications [24, 141, 100]).

7.1 MWGrid

The Medieval Warfare on the Grid project (MWGrid)[2] seeks into logistics of medieval war, the battle of Manzikert (modern Malazgirt, Turkey) in 1071 AD, between the Byzantine Empire and the Seljuk Turks. It addresses the problem of military logistics using agent

[1]MWGrid is one of the seven pilot research projects to funded by the AHRC-EPSRC-JISC e-Science Initiative. This is a collaborative project between The Institute of Archaeology and Antiquity (IAA, Professor Vince Gaffney) and the School of Computer Science (Dr Georgios Theodoropoulos) at the University of Birmingham

[2]`http://cs.bham.ac.uk/research/projects/mwgrid`

based modelling and distributed simulations. In medieval states the need to collect and distribute resources to maintain armies affected all aspects of the political organisation of the state[24]. The defeat of the Byzantine army by the Seljuk Turks, and the following civil war, resulted in the collapse of Byzantine power in central Anatolia. However, historical research towards distribution of resources by military is concluded based on available documents and constrained analytical methods. In MWGrid, the emphasis is on modelling military logistics such as route paths of armies, their interaction and behaviour using agent based modelling. The designers are interested in the key events associated with the Byzantine army's march across Anatolia to reach Battle of Manzikert in 1071 AD, travelling across 700 miles from Constantinople (modern Istanbul) to Manzikert (modern Malazgrit) just north of Lake Van depicted in figure 7.1.

Figure 7.1: Anatolia (taken from [100]).

Basically, the idea is to fill the gaps between several historical sources of the battle such as the size of the army (ranging from 200,000 to 1 million), their routing paths and their capacity to carry several resources such as horses, tons of food and so on across several miles. The project aims to use agent-based modelling to simulate several scenarios to understand and provide an insight to the historical key events of the march and end of the Byzantine empire.

7.1.1 MWGrid Framework

MWGrid project models agents as the members of the army ranging from commander to servant in a hierarchical fashion. Their common goal is to reach the destination to win the battle. Agents do possess some autonomous behaviour in making strategy decisions during their march. It requires several executions of the simulation model with varying sizes of armies, planning and resources to ascertain the outcome of the key events. So, the architecture consists of two parts namely, the simulation system and analysis environment as depicted in figure 7.2.

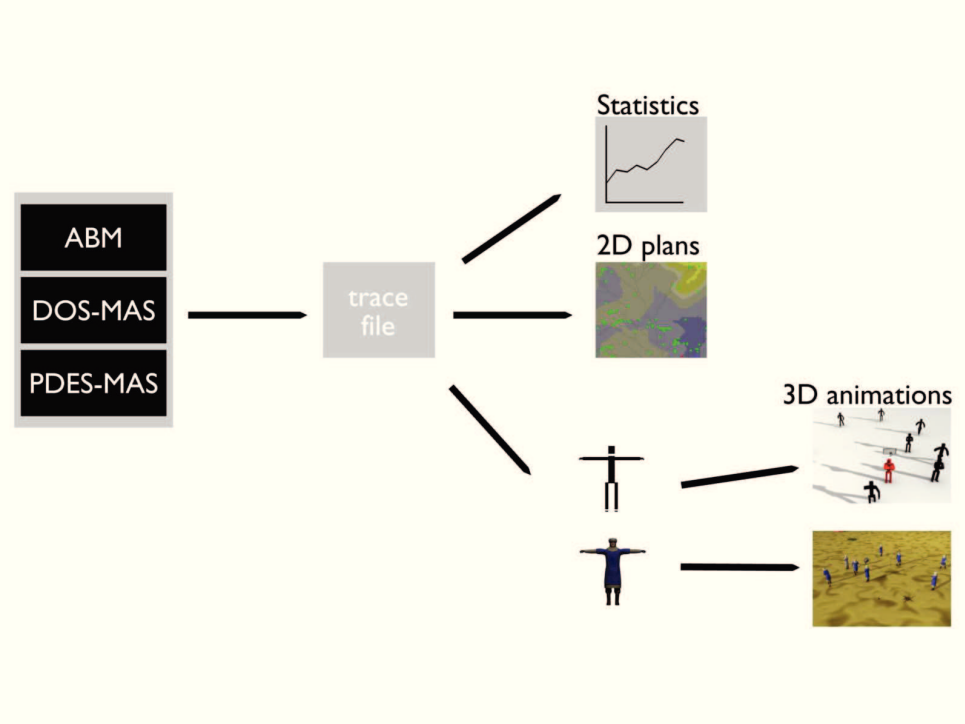


Figure 7.2: The MWGrid architecture (taken from [100]).

The simulation system executes the simulation model and produces a detailed output analysis of the framework using trace files. These trace files are then used to produce statistics detailing movement rates, food consumption, agent health status, time spent on movement and state of environment [100]. It can also be used to produce 3D visualisation of the simulation. To enable such functions, the simulation is divided into three layers such as

1. The Agent-based Model (ABM),

2. The distributed simulation kernel (PDES-MAS) and

3. The Middleware (DOS-MAS), which glues PDES-MAS with ABM.

The complete design details of the simulation model and its interaction model is discussed in [100]. As a summary, the simulation models agents (using human and animal behaviours) and environment (using terrain, infrastructure and resources of Anatolia). The data for the environment, though not accurate, has been populated from several resources and facts about the terrain, water and food resources and weather. On the other hand, the agents have several functionalities (mostly based on rank of the member) and each agent is identified with a unique identifier (ID) and rank. An agent can have both private and public variables. They communicate with each other using messages and aware of their hierarchical structure. For example, a commander may issue an order to a group of army members through messages using their ranks. Also, some agents possess abilities to plan and replan to reach the destination. For example, low level soldiers don't posses such ability and simply flock with their neighbours. A 3D visualisation of the environment of MWGrid model having soldiers and their camp sites is depicted in figure 7.3. The architecture of an agent and the environment is explained in detail in [100].

7.1.2 PDES-MAS framework

PDES-MAS framework (as discussed in detail in chapter 3) provides a distributed architecture to distribute and run parallel simulation of agent based models such as MWGrid. PDES-MAS is a PDES simulation kernel with optimistic synchronisation approach to synchronise the events according to LCC. Such property is useful for autonomous and dynamic agent based model such as MWGrid. MWGrid perfectly fits in PDES-MAS as agents are based on 'sense-think-act' cycle and agents interact with the environment to get the required information. Here, the environment is modelled as shared state of the simulation system and a tree of CLPs maintain the state. The vision range of an agent

Figure 7.3: A 3D visualisation of MWGrid simulation model.

is modelled as range queries and they are served in a time synchronised manner using algorithms presented in chapter 4. PDES-MAS also provides an adaptive approach to migrate a portion of shared state across the CLPs to reduce access latency and simulation time (discussed in detail in chapter 5). As a known fact of PDES-MAS is that it does not make use of proxies and so all information are accessed from CLPs using query mechanisms such as ID query and Range Query.

7.2 Middleware for MWGrid

PDES-MAS currently runs distributed simulation with agent traces and XML files (storing initial state of simulation such as SSV IDs, their initial values, types and so on as discussed in chapter 3) generated from MAS simulations. Currently, an extensive work and effort has been made to design a middleware to address the issue of interfacing a MAS with PDES-MAS framework. The idea is that middleware layer acts as a glue between PDES-MAS

framework and MAS such that shared state is distributed using PDES-MAS framework and run MAS simulation. A detailed overview of middleware architecture is presented in [24]. The figure 7.4 depicts the middlware architecture.

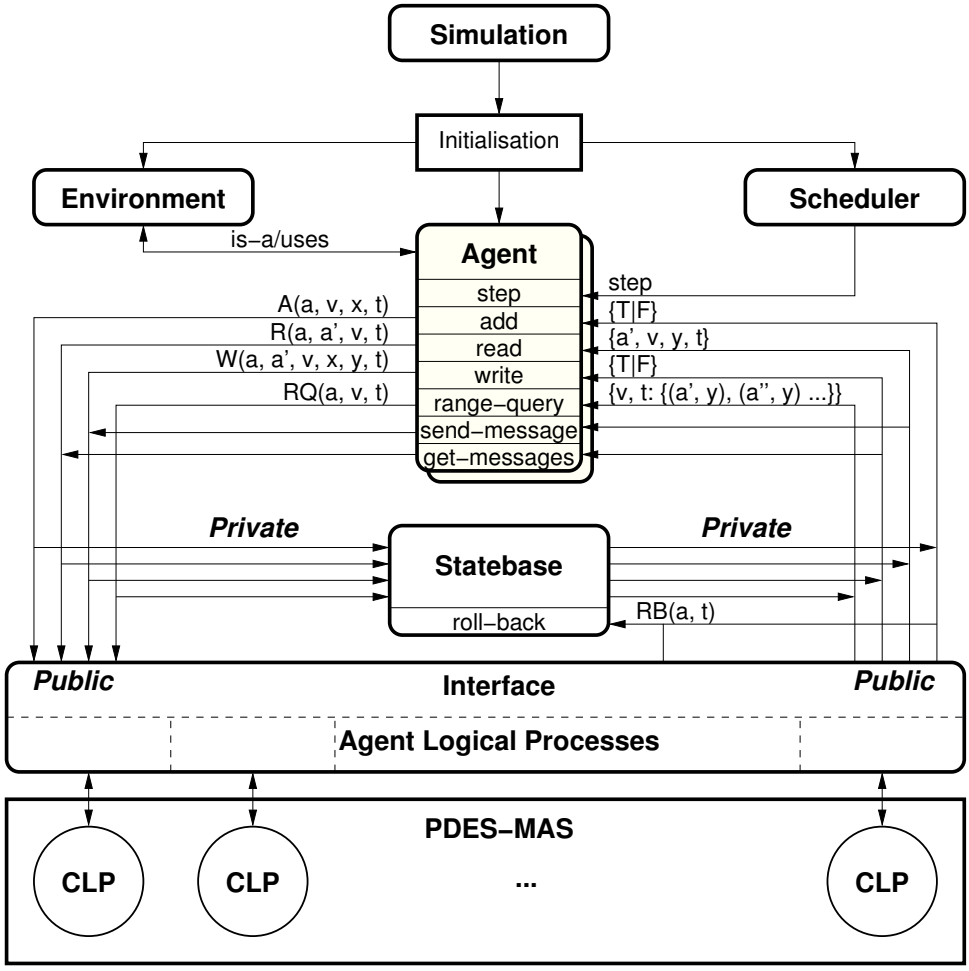


Figure 7.4: The middleware architecture (taken from [24]).

The Middleware implements a *Distributed Object Model (DOM)* and the *model layer* with basic templates and abstract objects to use and as well as an interface to access shared state variables from PDES-MAS framework. The middleware provides mechanisms to convert events from MAS simulation into format understandable in PDES-MAS and vice-versa. It also provides functionalities to initialise, start-up and gather output state to and from the simulation. The important aspect of this middleware is to provide mechanisms to handle rollbacks generated from PDES-MAS to rollback state in MAS itself. The middleware masks the existence of PDES-MAS and acts as an interface to

which agents can communicate with each other. The agent communication is implemented within PDES-MAS as a mailbox system where an agent can send a message to another agent with a timestamp. MWGrid and Middleware are implemented in JAVA and PDES-MAS framework in C++ programming languages respectively. The communication between PDES-MAS with MWGrid is established with a JAVA-JNI interface layer. I have contributed towards the design and development of the interface layer. I have also been involved with experimentation of the system on a distributed environment. Initial analysis presented in [24] suggest that there is a less overhead and seamless transmission of events between MAS and middleware and PDES-MAS architecture.

7.3 Experimental Analysis

The experimental investigation focusses on modelling key events of the Battle of Manzikert in 1071 AD under different conditions and scenarios. It involves different configurations of PDES-MAS architecture and MWGrid simulation model. At the time of writing of this thesis, common parameters involved in MWGrid simulation are number of agents/squads (size of army), miles covered (marching distance) and simulation ticks (typically covering a day or several days). Within PDES-MAS framework, the architecture is varied with Number of CLPs and Number of ALPs to understand the effect of overloading nodes with several agents and their accesses. It is currently on the working stage but an initial analysis involving several agents varying from 10 to 2000 agents have provided an insight to the performance of the PDES-MAS simulation kernel. The performance is measured using simulation time of experimental run using two test setups namely

1. Test Setup – 1 has Number of Agents varying from 10 to 1000, Number of ALPs = 4 and Number of CLPs = 3

2. Test Setup – 2 has Number of Agents varying from 10 to 1000, Number of ALPs = 8 and Number of CLPs = 7 and

The figure 7.5 presents the simulation time for above mentioned experimental setups. Typically, the time increases linearly with increasing Number of Agents. An interesting observation is that if we compare both setups (the difference being Number of ALPs and Number of CLPs), as we increase Number of Agents, test setup – 2 outperforms setup – 1 in terms of simulation time as it reflects the load incurred on the CLP tree. In this case, the higher depth with Test Setup – 2 distributes the shared state much better than Test Setup – 1 as the load increases with Number of ALPs.

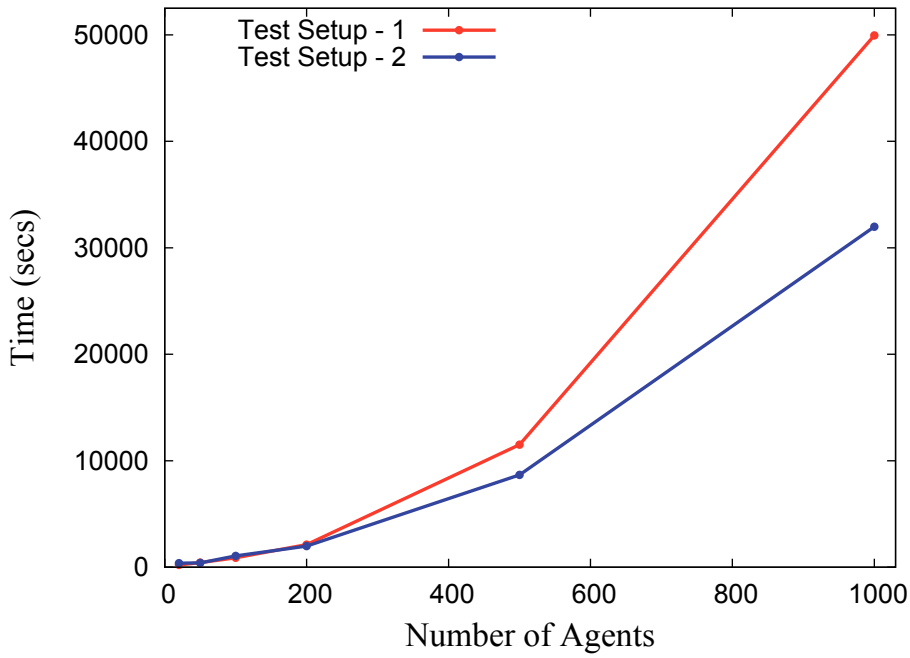


Figure 7.5: Simulation Time with different experimental setups of MWGrid model.

However, the work is still in progress to run experiments with hundreds of thousands of agents with varying depths of Number of CLPs and Number of ALPs. But the results show that PDES-MAS framework integrated with all algorithms presented in chapters 4 and 5 is workable and useful for simulating large scale agent based models.

7.4 Summary

In the MWGrid project the Middleware (see section 7.2) and PDES-MAS framework (presented in chapters 3, 4 and 5) combination was used to design and build a MAS for modelling the implications for pre-industrial societies of marching a large medieval army across the breadth of the Anatolian mainland of the Byzantine empire. Agents in the MAS represent all participants of the army from the emperor down to the individual soldier, all by extending the agent template provided by the Middleware. Agent and environment attributes and variables are based on detailed historical and geological analysis (see [100] for more details of the model). The MAS based on the Middleware and PDES-MAS framework combination allows running what-if scenarios with different configurations and, for example, army sizes. The results of these experiments are expected to have significant implications for the study of pre-industrial societies in methodological and theoretical terms, and will benefit academics with an interest in comparative military history, the cultural role of military organisation and the relationship of historical and modelled data[24]. Though initial analysis is not significant, we have pointed out that PDES-MAS framework (integrated with algorithms of synchronised range queries and state migration) is useful for agent based models. We have also shown that the interface created between PDES-MAS and MW-Grid model is workable.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This thesis presented an instantaneous range queries accessed in a time synchronised manner implemented within PDES-MAS framework. PDES-MAS is a simulation kernel to distribute and run parallel simulation of MAS. PDES-MAS is implemented based on PDES paradigm, where agents in MAS are modelled as Agent Logical Processes (ALPs) and shared state is maintained by a set of Communciation Logical Processes (CLPs). It is built on a tree structured overlay in which CLPs form intermediate nodes and ALPs are attached as leaves. ALPs use query based mechanisms to access shared variables from CLPs. The notion of a range query is used to access a set of variables that match a range predicate. The action simulates the sensing capabilities of agents. In this context, a range query issued over a CLP tree creates a horizon in which all SSVs within the horizon are scanned and SSVs outside the horizon are not. Changes in the values of SSVs within the horizon creates rollbacks directly and outside horizon creates rollbacks indirectly through range updates. Following PDES paradigm, values of SSVs are maintained in time periods at different ports in a CLP. Within this context, distributed data structures are implemented to maintain range periods (validity period of a range with a start and end time) at different time periods at different ports. However, to localise data access, state migration algorithm migrates a portion of shared state towards the agents that access the most. The extra cost and time incurred with migration algorithm is compensated with localised data access reducing access latency and simulation time. Synchronised range

queries and state migration algorithms implemented within this framework have shown to improve the performance of the system.

8.1 Summary of Results

Experimental investigation has been carried out at different stages in this thesis. A set of experiments evaluated the working of the algorithms (Synchronised Range Queries and State Migration) with predictable simulation parameters as presented in chapters 4 and 5. The purpose of such evaluation is to understand the behaviour of the algorithms under different conditions of the simulation. The results showed the relation between the experimental parameters and the outcome of results. These evaluations used synthetic traces generated using various simulation toolkits such as MWGrid. The analysis has also pointed out the overhead of state migration algorithm in terms of rollbacks and its effect on simulation time. The evaluation of the system with a benchmark analysis with agent based simulation toolkits such as TileWorld and Boids are presented in section 6.2 within chapter 6. The analysis has shown that the performance of the system (measured with access cost and time) has improved substantially. The scalability analysis (in section 6.3 within chapter 6), with varying Number of ALPs and Number of CLPs, has also showed evidences of bandwidth reduction with fewer rollbacks and localised data access. The final part of the thesis covered an overview of the design and implementation of MWGrid simulation as a use-case/benchmark for the evaluation of PDES-MAS simulation kernel. It is still in working stage but, at the time of writing, the initial analysis has shown promises towards the direction of large scale experimentation.

8.1.1 Synchronised Range Queries

A Range Query in this system is an ALP request for set of variable IDs with values that match a range predicate, which is a typical (min,max) range of values. To make sure that each ALP receives a set of variables consistently and accurately in a time synchronised

manner,

1. We have provided data structures to maintain a range information (a typical (min, max)) of values of the variables at each port of a CLP. Following PDES paradigm, values of SSVs are maintained for time periods. So, range periods data structure maintain range information at different time periods reflecting changes in the values of variables at different time periods of the simulation.

2. We have provided algorithms to rollback a premature (in the sense a read arrived in real time earlier with time > straggler write time) range query and delete its existence in the tree synchronously.

3. We have also provided mechanisms to update changes in values of SSVs to all external ports using time synchronised range updates.

The experimental evaluation focussed on analysing the performance over various metrics of the concurrent processes of: Range Query propagation; Synchronisation via Range Query Rollback; and Range Update propagation in response to write/anti-write patterns. First experiment focussed on the impact of two parameters (*RQ size* and *SSV range*) on extent of range queries over the CLP tree. The results showed that

1. The relation between experimental parameters such as *RQ size* and *SSV range* and range query extent is evident on the results. On average, a range query traversed almost half the depth of CLP tree (7 Clps tree, in this case) with maximum *RQ size* and *SSV range* values which is half the proportional size of simulation world.

2. The query response time (time it takes to process a range query to get back with its response to the querying ALP) also follows a similar pattern with different experimental parameters. It 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).

The second part of experimental analysis focussed on the relationship between Write operations (changes in values of SSVs) and the consequent rate at which Range Updates

(changes in range information) occur. Two parameters are used *SSV per CLP* and *Write Delta*. Predictably, range update rate has a linear relationship with the parameters. It increased from 0 (with *Write Delta* = 0 and *SSV per CLP* = 2) to 0.5 (with *Write Delta* = 8 and *SSV per CLP* = 2). Obviously when *SSV per CLP* increases, the rate decreased linearly.

The final experiment focussed on rollback rate and wall clock (simulation execution) time when range queries (*RQ size*) and updates (*Write Delta*) are issued concurrently. A direct correlation between rollback rate and wall clock time and query and update patterns has been observed. It showed how volatility of updates affected rollback rate and execution times. The rollback rate varied from 0.02 to 0.1 and simulation time varied from 49 secs to 98 secs.

8.1.2 State Migration

The adaptive approach of state migration algorithm localises data access by moving a portion of shared state across the CLP tree towards the ALPs that access the most. Localised data access reduces the extent of a range query thereby reducing access cost and simulation time of the system. For that purpose,

1. We have provided data structures at each CLP, which internally maintains access and load information of state variables at each port.

2. We have provided mechanisms to migrate state variables without affecting the flow of transient messages in the system.

3. We also have provided mechanisms to update the changes such as locations of state variables and range information across the system.

The experimental investigation focussed on the impact of range query extent and state migration on access cost of the simulation and rollback volatility on simulation time. All SSVs are placed at root CLPs to clearly show the effect of state migration. The results showed that

1. The propagation (extent) of a range query is reduced With SM which ranges from 1.4 to almost 3 with increasing *RQ size* and *SSV range* parameter values (in comparison to 7 Without SM). This reduced the access cost on average 47.12% for all values of *RQ size* and *SSV range*. It showed the extra cost incurred with migration is compensated with reduction in range query propagation.

2. On the contrary, the volatility of rollbacks affected the performance of the system with small reduction in simulation time until *RQ size* reaches 0.3 and time With SM increases more than Without SM. The evaluation is carried out using random traces (which in this case turned out be worst case) and though the results are not promising, it showed the correlation between rollbacks and simulation time.

8.2 PDES-MAS

PDES-MAS simulation kernel has been designed and developed as a large scale distributed simulation system capable of running a complex system such as MAS. The performance of PDES-MAS simulation kernel is measured with a benchmark analysis using traces of simulation test beds such as TileWorld and Boids. The first part of this investigation measured the performance using three metrics: Query propagation, access cost and simulation time. TileWorld is highly dynamic with volatile movements of agents in the environment with higher possibilities of rollbacks and migrations. The evaluations of the kernel using TileWorld simulation showed that

1. The query propagation is reduced linearly with different values of *RQ size*, *Creation Probability* and *Environment Size* parameter values. It is evident that migration algorithm reduced query propagation on average to 5.4 and update propagation to 2.8 (compared to 7 Without SM).

2. The impact of extra cost of initiating migration algorithm is compensated with localised data access which is evident from the fact that the access cost is reduced

on average to 66.46% and simulation time is reduced by 25.94%.

On the other hand, Boids is a collaborative agent based modelling used in understanding flocking or herding behaviours. The evaluation using Boids simulation showed that

1. The query propagation is reduced linearly with different values of *RQ size* and *Velocity* parameter values. Query propagation is reduced to 5.5, whereas update propagation is reduced to just 1. This has huge impact on access cost which is reduced by 30%.

2. Though rollback volatility is not significant in this simulation (as there are few rollbacks initiated) but the impact of each rollback in terms of its depth is quite high. Though the depth is even higher With SM, the effect is diminished with localised data access that reduces simulation time by 11%.

Second part of evaluation analysed the scalability of PDES-MAS simulation kernel within available hardware resources. The idea is to increase the number of agents connected to the system linearly and performance evaluation focussed on simulation times. Also, Number of CLPs is increased linearly to analyse the impact of dynamic distribution of shared state on the performance. The trade-off is between minimising traffic and access times and computations of migration mechanisms. The results are compared with $1CLP$ tree configuration to show the significance of scalability. It showed that

1. With smaller configuration of Number of CLPs and Number of ALPs, the migration algorithm out performs in terms of simulation times in comparison with Number of CLPs = 1. However, increasing Number of ALPs with minimum Number of CLPs increases bottleneck of handling queries concurrently. There is an exponential increase in query response and simulation times.

2. However, as both Number of ALPs and Number of CLPs increase linearly, the performance improved until Number of CLPs reaches 15. When the depth of CLP

tree is increased to the maximum, computation and time required to migrate SSVs closer to the agents become an overhead.

The experimental setup used for this evaluation may not be adequate enough to show PDES-MAS simulation kernel is part of very large scale distributed systems but there are evidences in the results that point towards that direction. Within available resources, the system has been tested to the maximum. In the future, with the availability of more resources and adaptive approaches and different network overlays the system can be tested with thousands or even hundreds of thousands of agent based modelling.

8.3 MWGrid: Use-Case

MWGrid (Medieval Warfare on the Grid) project seeks to address the problems of military logistics of the Battle of Manzikert in 1071 AD using agent based modelling and distributed simulations. PDES-MAS provides a distributed simulation architecture and a Middleware framework acting as a glue between PDES-MAS and the MAS simulation (MWGrid). Basically, the idea of Middleware framework is to provide a light-weight solution to translate events to and from MAS simulation into the format of PDES-MAS framework. Considerable effort in design and implementation has been made to build a system that is capable of handling large scale experimentation involving hundreds of thousands of agents. However, the project is still in working stage but an initial analysis of the system showed promises towards that direction. In the future, such systems will provide a good benchmark to embark on developing even more complex and dynamic simulation models.

8.4 Future Experiments and Development

The analysis of PDES-MAS simulation kernel suggests that there are significant and substantial improvements required to simulate hundreds of thousand of agents or may

be in millions. With an optimistic approach to synchronise events, PDES-MAS will always suffer from rollback computations. Though localised data access reduces or masks such severities, it is not adequate enough to run a scalable system. Attempts have been made towards adaptive synchronisation approaches for PDES-MAS system and is reported in [72, 73, 65, 64, 66, 74, 75]. However, such algorithms are tested with a small tree configuration of 1 CLP with no distribution mechanism. Also, PDES-MAS system have been ported with synchronised range queries and migration algorithms. Thus, it requires a slightly different approach to address the issue. But it would be a good benchmark analysis to test such adaptive approaches with the existing PDES-MAS system.

Another aspect of PDES-MAS kernel system is the requirement of a more random and de-centralised approach to distribute shared data across the nodes. This suggests that peer-to-peer overlays can be ported over existing binary tree or a different structure to maintain shared data. This will improve distribution much significantly and also the routing mechanism. It also allows to add and delete variables from the nodes easily with less requirement for explicit load balancing mechanisms. However, it would be interesting to analyse the performance of range queries over such network overlays as generally DHTs are not suitable for such mechanism.

The evaluation of PDES-MAS is measured with traces generated from different MAS. The interface developed within PDES-MAS is sufficient enough to port traces of multiple agents running in parallel. Experimental evaluation provides a good benchmark to design and test different architecture. However, it is not possible to visualise the working of PDES-MAS system or the impact of rollbacks on an agent based modelling. The implementation of Middleware framework as reported in [24] bridges the gap between PDES-MAS framework and complex MAS simulation such as MWGrid [100]. However, it is still in working stage and considerable effort is required for visualisation and large scale experimentation.

LIST OF REFERENCES

[1] (January 2007) Gnutella. [Online]. Available: http://www.gnutella.com/

[2] E. A.Alluisi, "The development of technology for collective training: Simnet, a case history," *Hum. Factors*, vol. 33, pp. 343–362, May 1991.

[3] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in charon," in *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, ser. HSCC '00. London, UK: Springer-Verlag, 2000, pp. 6–19.

[4] J. Anderson, "A generic distributed simulation system for intelligent agent design and evaluation," *Proceedings of the 10th International Conference on AI Simulation and Planning in High Autonomy Systems*, pp. 36–44, 2000.

[5] J. Aspnes and G. Shah, "Skip graphs," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '03. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003, pp. 384–393.

[6] H. Avril and C. Tropper, "Clustered time warp and logic simulation," in *Proceedings of the ninth workshop on Parallel and distributed simulation*, ser. PADS '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 112–119.

[7] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting large multiuser virtual environments," *IEEE Computer Graphics and Applications*, vol. 16, pp. 50–57, 1996.

[8] S. Benford and L. Fahlén, "A spatial model of interaction in large virtual environments," in *Proceedings of the third conference on European Conference on Computer-Supported Cooperative Work*. Norwell, MA, USA: Kluwer Academic Publishers, 1993, pp. 109–124.

[9] S. Benford and C. Greenhalgh, "Introducing third party objects into the spatial model of interaction," in *Proceedings of the fifth conference on European Conference on Computer-Supported Cooperative Work*. Norwell, MA, USA: Kluwer Academic Publishers, 1997, pp. 189–204.

[10] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and A. deKorvin, "Approaches to multicast group allocation in hla data distribution management," in *In Proceedings of the 1998 Spring Simulation Interoperability Workshop*, 1998.

[11] A. Bharambe, "Colyseus: A distributed architecture for online multiplayer games," in *In Proc. Symposium on Networked Systems Design and Implementation (NSDI*, 2006, pp. 3–06.

[12] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 353–366, 2004.

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

[14] D. Black, A. Gupta, and W. D. Weber, "Competitive management of distributed shared memory," in *COMPCON*, 1989.

[15] Blizzard-Entertainment. (March 2006) World of warcraft. [Online]. Available: http://www.worldofwarcraft.com/

[16] A. Boukerche and A. Roy, "Dynamic grid-based approach to data distribution management," *J. Parallel Distrib. Comput.*, vol. 62, no. 3, pp. 366–392, 2002.

[17] A. Boukerche and C. Tropper, "A static partitioning and mapping algorithm for conservative parallel simulations," in *Proceedings of the eighth workshop on Parallel and distributed simulation*, ser. PADS '94. New York, NY, USA: ACM, 1994, pp. 164–172.

[18] C. Burdorf and J. Marti, "Load balancing strategies for time warp on multi-user workstations," *Comput. J.*, vol. 36, no. 2, pp. 168–176, 1993.

[19] C. Carlsson and O. Hagsand, "Dive - a multi user virtual reality system." in *VR9́3*, 1993, pp. 394–400.

[20] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *In Infocom03*, 2003.

[21] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM*, vol. 24, pp. 198–206, April 1981.

[22] K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Trans. Comput. Syst.*, vol. 3, pp. 63–75, February 1985.

[23] T.-C. Chiueh, "Distributed systems support for networked games," *Hot Topics in Operating Systems, Workshop on*, p. 99, 1997.

[24] B. Craenen, V. Suryanarayanan, and G. Theodoropoulos, "A middleware for interfacing with simulation systems of multi-agent models." ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), March, 2011, barcelona, Spain.

[25] R. C.Waters, D. B.Anderson, J. W.Barrus, D. C.Brogan, M. A.Casey, S. G.Mckeown, T. Nitta, I. B.Sterns, and W. S.Yerazunis, "Diamond park and spline: Social virtual reality with 3d animation, spoken interaction, and runtime extendability," *Teleoperators and Virtual Environments*, vol. 6, pp. 461–481, 1997.

[26] S. R. Das, "Estimating the cost of throttled execution in time warp," in *Proceedings of the tenth workshop on Parallel and distributed simulation*, ser. PADS '96. Washington, DC, USA: IEEE Computer Society, 1996, pp. 186–189.

[27] S. R. Das and R. M. Fujimoto, "Adaptive memory management and optimism control in time warp," *ACM Trans. Model. Comput. Simul.*, vol. 7, pp. 239–271, April 1997. [Online]. Available: http://doi.acm.org/10.1145/249204.249207

[28] S. R. Das and R. Fujimoto, "An empirical evaluation of performance-memory trade-offs in time warp," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, pp. 210–224, 1997.

[29] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer, "Range queries in trie-structured overlays," in *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 57–66.

[30] S. J. Eggers and R. H. Katz, "Evaluating the performance of four snooping cache coherency protocols," *SIGARCH Comput. Archit. News*, vol. 17, no. 3, pp. 2–15, 1989.

[31] R. Ewald, D. Chen, G. K. Theodoropoulos, M. Lees, B. Logan, T. Oguara, and A. M. Uhrmacher, "Performance analysis of shared data access algorithms for distributed simulation of multi-agent systems," in *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 29–36.

[32] J. Fanning and S. Fanning. (June 1999) Napster. [Online]. Available: http://www.napster.com/

[33] A. Ferscha and G. Chiola, "Self-adaptive logical processes: the probabilistic distributed simulation protocol," in *In Proc. of the 27 th Annual Simulation Symposium*. IEEE Computer Society Press, 1994, pp. 78–88.

[34] A. Ferscha and J. Luthi, "Estimating rollback overhead for optimism control in time warp," in *Proceedings of the 28th Annual Simulation Symposium*, ser. SS '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 2–.

[35] A. Ferscha and J. Johnson, "Shock resistant time warp," in *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, ser. PADS '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 92–100.

[36] G. S. Fishman, *Principles of Discrete Event Simulation*. New York, USA: John Wiley & Sons, Inc., 1978.

[37] R. M. Fujimoto, *Parallel and Distributed Simulation systems*. New York, NY, USA: John Wiley & Sons Inc., 2000.

[38] Fujitsu. (June 2011) Fujitsu k computer. [Online]. Available: http://www.lanl.gov/roadrunner/

[39] T. A. Funkhouser, "Ring: A client-server system for multi-user virtual environments," in *Symposium on Interactive 3D Graphics*, 1995, pp. 85–92.

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

[41] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 1, pp. 80–112, 1985.

[42] N. Gilbert and S. Bankes, "Platforms and methods for agent-based modeling," *Proceedings of The National Academy of Sciences*, vol. 99, pp. 7197–7198, 2002.

[43] D. W. Glazer and C. Tropper, "On process migration and load balancing in time warp," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, pp. 318–327, March 1993.

[44] C. Greenhalgh and S. Benford, "Massive: a collaborative virtual environment for teleconferencing," *ACM Trans. Comput.-Hum. Interact.*, vol. 2, pp. 239–261, September 1995.

[45] S. A. Gupta, A. Gupta, D. Agrawal, and A. E. Abbadi, "Approximate range selection queries in peer-to-peer," in *In CIDR*, 2002.

[46] D. O. Hamnes and A. Tripathi, "Investigations in adaptive distributed simulation," in *Proceedings of the eighth workshop on Parallel and distributed simulation*, ser. PADS '94. New York, NY, USA: ACM, 1994, pp. 20–23.

[47] R. C. Hofer and M. L. Loper, "Dis today [distributed interactive simulation]," *Proceedings of The IEEE*, vol. 83, pp. 1124–1137, 1995.

[48] D. V. Hook, S. J. Rak, and J. O. Calvin, "Approaches to relevance filtering," in *In Eleventh Workshop on Standards for the Interoperability of Distributed Simulations*, 1994, pp. 26–30.

[49] C. Horstmann, "Gridworld," September 2008, http://www.horstmann.com/gridworld/.

[50] S.-Y. Hu and G.-M. Lia, "Scalable peer-to-peer networked virtual environment," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, ser. NetGames '04. New York, NY, USA: ACM, 2004, pp. 129–133.

[51] M. Hyett and R. Wuerfel, "Implementation of the data distribution management services in the rti-ng," in *In Proceedings of the 2001 Fall Simulation Interoperability Workshop*, 2001.

[52] IBM. (May 2008) Ibm roadrunner. [Online]. Available: http://www.lanl.gov/roadrunner/

[53] id Software. (June 1996) Quake. [Online]. Available: http://www.QuakeLive.com/

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

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

[56] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, pp. 404–425, July 1985.

[57] N. R. Jennings and M. J. Wooldridge, Eds., *Agent Technology: Foundations, Applications, Markets*. Springer-Verlag, 1998, ch. Applications of Intelligent Agents, pp. 3–28.

[58] K. L. Kapp, T. C. Hartrum, and T. S. Wailes, "An improved cost function for static partitioning of parallel circuit simulations using a conservative synchronization protocol," in *Proceedings of the ninth workshop on Parallel and distributed simulation*, ser. PADS '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 78–85.

[59] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching." *Algorithmica*, pp. 77–119, 1988.

[60] E. Keong, L. J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, pp. 72–93, 2005.

[61] B. Knutsson, M. M. Games, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *In Proceedings of INFOCOMM 2004*, March 2004.

[62] L. Lamport, "Specifying concurrent program modules," *ACM Trans. Program. Lang. Syst.*, vol. 5, pp. 190–222, April 1983.

[63] R. Lea, Y. Honda, and K. Matsuda, "Virtual society: Collaboration in 3d spaces on the internet," *Comput. Supported Coop. Work*, vol. 6, pp. 227–250, May 1997.

[64] M. Lees, B. Logan, C. Dan, T. Oguara, and G. Theodoropoulos, "Analysing probabilistically constrained optimism," pp. 201–208.

[65] M. Lees, B. Logan, C. Dan, T. Oguara, and G. Theodoropoulos, "Decision-theoretic throttling for optimistic simulations of multi-agent systems," in *Proceedings of the Ninth IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2005)*, A. Boukerche, S. J. Turner, D. Roberts, and G. Theodoropoulos, Eds. Montreal, Quebec, Canada: IEEE Press, October 2005, pp. 171–178.

[66] M. Lees, B. Logan, C. Dan, T. Oguara, and G. Theodoropoulos, "Analysing the performance of optimistic synchronisation algorithms in simulations of multi-agent systems," in *PADS '06: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 37–44.

[67] M. Lees, B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos, "Distributed simulation of MAS," in *Multi-Agent and Multi-Agent-Based Simulation: Joint Workshop MABS 2004*, ser. LNAI, P. Davidsson, B. Logan, and K. Takadama, Eds., no. 3415. Springer, 2004, pp. 25–36.

[68] M. Lees, B. Logan, R. Minson, T. Oguara, and G. Theodoropoulos, "Modelling environments for distributed simulation," in *Environments for Multi-Agent Systems: Proceedings of the the First International Workshop (E4MAS'04)*, ser. LNAI, D. Weyns, H. V. D. Parunak, and F. Michel, Eds., no. 3374. Springer, 2004, pp. 150–167.

[69] M. Lees, B. Logan, T. Oguara, and G. Theodoropoulos, "Simulating agent-based systems with HLA: The case of SIM_AGENT – part II," in *Proceedings of the 2003 European Simulation Interoperability Workshop*, European Office of Aerospace R&D. Simulation Interoperability Standards Organisation and Society for Computer Simulation International, June 2003.

[70] M. Lees, B. Logan, T. Oguara, and G. Theodoropoulos, "Hla_agent: Distributed simulation of agent-based systems with hla," in *In Proc. of the International Conference on Computational Science (ICCS'04*. Springer, 2004, pp. 907–915.

[71] M. Lees, B. Logan, and G. Theodoropoulos, "Simulating agent-based systems with HLA: The case of SIM_AGENT," in *Proceedings of the 2002 European Simulation Interoperability Workshop*, European Office of Aerospace R&D. Simulation Interoperability Standards Organisation and Society for Computer Simulation International, June 2002, pp. 285–293.

[72] M. Lees, B. Logan, and G. Theodoropoulos, "Adaptive optimistic synchronisation for multi-agent simulation," in *17th European Simulation Multiconference (ESM'03)*, 2003, pp. 77–82.

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

[74] M. Lees, B. Logan, and G. Theodoropoulos, "Using access patterns to analyze the performance of optimistic synchronization algorithms in simulations of mas," *Simulation*, vol. 84, no. 10/11, pp. 481–492, October 2008.

[75] M. Lees, B. Logan, and G. Theodoropoulos, "Analysing probabilistically constrained optimism," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 11, pp. 1467–1482, August 2009.

[76] L.Gautier and C.Diot, "Design and evaluation of mimaze, a multi-player game on the internet," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, ser. ICMCS '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 233–.

[77] K. Li and P. Hudak, "Memory coherence in shared virtual memory systems," *ACM Trans. Comput. Syst.*, vol. 7, pp. 321–359, November 1989.

[78] J. Liang, R. Kumar, and K. W. Ross, "The kazaa overlay: A measurement study," *Computer Networks Journal (Elsevier*, 2005.

[79] B. Logan and G. Theodoropoulos, "The Distributed Simulation of Multi-Agent Systems," vol. 89, no. 2, Feb 2001, pp. 174–186.

[80] B. Lubachevsky, A. Shwartz, and A. Weiss, "Rollback sometimes works...if filtered," in *Proceedings of the 21st conference on Winter simulation*, ser. WSC '89. New York, NY, USA: ACM, 1989, pp. 630–639.

[81] B. Lubachevsky, A. Weiss, and A. Shwartz, "An analysis of rollback-based simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 1, pp. 154–193, 1991.

[82] S. Luke, C. Cioffi-revilla, L. Panait, and K. Sullivan, "Mason: A new multi-agent simulation toolkit," in *University of Michigan*, 2004.

[83] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "Npsnet: A network software architecture for large scale virtual environments," *Presence*, vol. 3, pp. 265–287, 1994.

[84] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups," *IEEE Comput. Graph. Appl.*, vol. 15, no. 5, pp. 38–45, 1995.

[85] E. Mascarenhas, F. Knop, R. Pasquini, and V. Rego, "Minimum cost adaptive synchronization: experiments with the parasol system," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 401–430, October 1998.

[86] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[87] D. E. M. T. J. McBrayer and P. A. Wilsey, "Warped: Time warp simulation kernel for analysis and application development." in *HICSS (1)'96*, 1996, pp. 383–386.

[88] R. Minson and G. Theodoropoulos, "Distributing repast agent-based simulations with hla," in *In: Proceedings of the 2004 European Simulation Interoperability Workshop, Edinburgh, Simulation Interoperability Standards Organisation and Society for Computer Simulation International*, 2004.

[89] R. Minson and G. Theodoropoulos, "An adaptive interest management scheme for distributed virtual environments," in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 273–281.

[90] R. Minson and G. Theodoropoulos, "Adaptive interest management via push-pull algorithms," in *DS-RT '06: Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 119–126.

[91] R. Minson and G. Theodoropoulos, "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*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 53–60.

[92] R. Minson and G. Theodoropoulos, "Push-pull interest management for virtual worlds," in *ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 189–194.

[93] R. C. Minson, "Adaptive push-pull algorithms for distributed virtual environments," Ph.D. dissertation, Computer Science Department, 2010.

[94] G. Morgan, K. Storey, and F. Lu, "Expanding spheres: A collision detection algorithm for interest management in networked games," in *ICEC*, 2004, pp. 435–440.

[95] K. L. Morse, "Interest management in large-scale distributed simulations," in *Virtual Reality*, 1996.

[96] K. L. Morse and M. D. Petty, "High level architecture data distribution management migration from dod 1.3 to ieee 1516," *Concurrency and Computation: Practice and Experience*, vol. 16, pp. 1527–1543, 2004.

[97] K. L. Morse and J. S. Steinman, "Data distribution management in the hla: Multidimensional regions and physically correct filtering," in *In Proceedings of the 1997 Spring Simulation Interoperability Workshop*. Spring, 1997, pp. 343–352.

[98] K. L. Morse and M. Zyda, "Multicast grouping for data distribution management," *Simul. Pr. Theory*, vol. 9, no. 3-5, pp. 121–141, 2002.

[99] K. L. Morse, "An adaptive, distributed algorithm for interest management," Ph.D. dissertation, Information and Computer Science, 2000, chair-Bic, Lubomir and Chair-Dillencourt, Michael.

[100] P. Murgatroyd, V. Gaffney, B. Craenen, G. Theodoropoulos, V. Suryanarayanan, and J. Haldon, "Logistics: A case of distributed agent-based simulation," in *Proceedings of the Distributed Simulation & Online Gaming Conference (DISIO 2010).* Torremolinos, Spain: ACM Digital Library, Mar 2010.

[101] B. Nandy and W. M. Loucks, "An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers," in *Workshop on Principles of Advanced and Distributed Simulation*, 1992, pp. 139–146.

[102] D. M. Nicol and P. F. Rcynolds, "A statistical approach to dynamic partitioning," in *Proceedings of the SCS Multi-Conference*, San Diego, 1985, pp. 53–56.

[103] M. J. North, N. T. Collier, and J. R. Vos, "Experiences creating three implementations of the repast agent modeling toolkit," *ACM Transactions on Modeling and Computer Simulation*, vol. 16, pp. 1–25, 2006.

[104] I. of Electrical and E. Engineers, "Ieee standard for distributed interactive simulationapplication protocols," *IEEE press*, 1995.

[105] T. Oguara, D. Chen, G. Theodoropoulos, B. Logan, and M. Lees, "An adaptive load management mechanism for distributed simulation of multi-agent systems," pp. 179–186.

[106] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer, "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.* New York, NY, USA: ACM, 1993, pp. 214–221.

[107] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, March 2006.

[108] A. C. Palaniswamy and P. A. Wilsey, "Adaptive bounded time windows in an optimistically synchronized simula-tor," in *ACM Great Lakes Symposium on VLSI*, 1993.

[109] K. S. Panesar and R. M. Fujimoto, "Adaptive flow control in time warp," *ACM Sigsim Simulation Digest*, vol. 27, pp. 108–115, 1997.

[110] S. R. Paul, F. M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *IN PROC. ACM SIGCOMM 2001*, 2001, pp. 161–172.

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

[112] S. Powers, M. Hinds, and J. Morphett, "Dee: an architecture for distributed virtual environment gaming." *Distributed Systems Engineering*, pp. 107–117, 1998.

[113] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Commun. ACM*, vol. 33, pp. 668–676, June 1990.

[114] J. Purbrick and C. Greenhalgh, "Extending locales: Awareness management in massive-3," in *Virtual Reality*, 2000.

[115] S. F. Railsback, L. Railsback, S. L. Lytinen, and S. K. Jackson, "Agent-based simulation platforms: review and development recommendations," in *ENVIRONMENTAL MODELLING & SOFTWARE 22 (2007) 1775E1787 RUSSEL, S.J., NORVIG, P.* Prentice-Hall, 2006, p. 609.

[116] S. J. Rak, M. Salisbury, and R. S. Macdonald, "Hla/rti data distribution management in the synthetic theater of war," in *In Proceedings of the Fall 1997 DIS Workshop on Simulation*, 1997.

[117] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker, "Brief announcement: prefix hash tree," in *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, ser. PODC '04. New York, NY, USA: ACM, 2004, pp. 368–368.

[118] P. L. Reiher and D. Jefferson, "Virtual time based dynamic load management in the time warp operating system," *Transactions of the Society for Computer Simulation*, vol. 7, pp. 103–111, 1990.

[119] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34.

[120] P. F. Reynolds, Jr., "A spectrum of options for parallel simulation," in *Proceedings of the 20th conference on Winter simulation*, ser. WSC '88. New York, NY, USA: ACM, 1988, pp. 325–332.

[121] P. Riley and G. F. Riley, "Next generation modeling iii - agents: Spades - a distributed agent simulation environment with software-in-the-loop execution," in *Winter Simulation Conference*, 2003, pp. 817–825.

[122] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: search methods," *Comput. Netw.*, vol. 50, pp. 3485–3521, December 2006.

[123] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[124] J. S. K. L. Russo and L. C. Schuette, "Prototype multicast ip implementation in modsaf," in *In 12th Workshop on Standards for the Interoperability of Distributed Simulations*, March 1995, pp. 175–178.

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

[126] F. Sarkar and S. K. Das, "Design and implementation of dynamic load balancing algorithms for rollback reduction in optimistic pdes," in *Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, 1997, pp. 26–31.

[127] R. Schlagenhaft, Martin, M. Ruhwandl, C. Sporrer, and H. Bauer, "Dynamic load balancing of a multi-cluster simulator on a network of workstations," in *In Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS*, 1995, pp. 175–180.

[128] A. Serenko and B. Detlor, "Agent toolkits: A general overview of the market and an assessment of instructor satisfaction with utilizing toolkits in the classroom," 2002, working Paper 455, McMaster University, Hamilton, Ontario, Canada.

[129] Skype-Limited. (August 2003) Skype website. [Online]. Available: http://www.skype.com/

[130] J. Smed, T. Kaukoranta, and H. Hakonen, "A review on networking and multiplayer computer games," in *IN MULTIPLAYER COMPUTER GAMES, PROC. INT. CONF. ON APPLICATION AND DEVELOPMENT OF COMPUTER GAMES IN THE 21ST CENTURY*, 2002, pp. 1–5.

[131] J. M. Squyres and A. Lumsdaine, "A Component Architecture for LAM/MPI," in *Proceedings, 10th European PVM/MPI Users' Group Meeting*, ser. Lecture Notes in Computer Science, no. 2840. Venice, Italy: Springer-Verlag, September / October 2003, pp. 379–387.

[132] S. Srinivasan and P. F. R. Jr., "Npsi adaptive synchronization algorithms for pdes," in *Winter Simulation Conference*, 1995, pp. 658–665.

[133] J. S.Steinman, "Interactive speedes," in *Proceedings of the 24th annual symposium on Simulation*, ser. ANSS '91. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 149–158.

[134] B. Stabell and K. R. Schouten, "The story of xpilot," *Crossroads*, vol. 3, pp. 3–6, November 1996.

[135] P. Stenström, "A survey of cache coherence schemes for multiprocessors," *Computer*, vol. 23, pp. 12–24, June 1990.

[136] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, 2001, pp. 149–160.

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

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

[139] S. C. Tay, Y. M. Teo, and S. T. Kong, "Speculative parallel simulation with an adaptive throttle scheme," in *Proceedings of the eleventh workshop on Parallel and distributed simulation*, ser. PADS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 116–123.

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

[141] G. Theodoropoulos, P. M. V. Gaffney, and B. Craenen, "Heading towards manzikert - the medieval warfare on the grid project," *Universitas 21 Digital Humanities Conference "Cultural Heritage and Technology"*, September, 2010.

[142] T. R. Tiernan, "Synthetic theater of war (stow) engineering demonstration-1 (ed-1) final report," 1996.

[143] C. Tropper and K. El-khatib, "On metrics for the dynamic load balancing of optimistic simulations," in *Hawaii International Conference on System Sciences*, 1999.

[144] A. I. T.Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, ser. Middleware '01. London, UK: Springer-Verlag, 2001, pp. 329–350.

[145] S. Turner and M. Qu, "Performance evaluation of the bounded time warp algorithm," in *Workshop on Principles of Advanced and Distributed Simulation*, 1992.

[146] P. T. Tymann and F. P. Trees, "Teaching with the gridworld case study," *J. Comput. Small Coll.*, vol. 23, pp. 118–119, May 2008.

[147] F. Wang, S. J. Turner, and L. Wang, "Integrating agents into hla-based distributed virtual environments," in *In Proc. of the Fourth Workshop on Agent-Based Simulation*, Montepellier, 2003.

[148] R. M. Weatherly, A. L.Wilson, and S. P.Griffin, "Alsptheory, experience, and future directions," in *Proceedings of the 25th conference on Winter simulation*, ser. WSC '93. New York, NY, USA: ACM, 1993, pp. 1068–1072.

[149] L. F. Wilson and W. Shen, "Experiments in load migration and dynamic load balancing in speedes," in *Winter Simulation Conference*, 1998, pp. 483–490.

[150] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.

[151] Yahoo. (March 1998) Yahoo messenger website. [Online]. Available: http://www.messenger.yahoo.com/

[152] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and," University of California at Berkeley, Berkeley, CA, USA, Tech. Rep., 2001.