INTEREST MANAGER MECHANISMS FOR DISTRIBUTED MULTI-AGENT-BASED SIMULATIONS USING AN OPTIMISTIC EXECUTIVE

by

TIANHAO HE

(Under the Direction of Maria Hybinette)

ABSTRACT

An important issue for agent-based simulation architectures concerns the means by which agents can be kept up to date regarding the position and state of other agents and objects in their local environment. We implement two distributed publish-subscribe approaches and assess their performance within the SASSY architecture. The environment is decomposed into a set of regions, where an LP referred to as an Interest Manager (IM) manages each region. Publish-subscribe approaches have been utilized in other simulation environments (e.g., HLA), but we believe our approach is the first to use a peer-to-peer method in an optimistic environment. We assess the efficiency and scalability of these two methods by measuring the number of rollbacks and total runtime for various configurations of the simulation. Further, we evaluate the impact of varying the disparity of between the sensor and actuator regions.

INDEX WORDS: Agent based simulation, Interest management, PDES, SASSY

INTEREST MANAGER MECHANISMS FOR DISTRIBUTED MULTI-AGENT-BASED SIMULATIONS USING AN OPTIMISTIC EXECUTIVE

by

TIANHAO HE

B.S, BEIJING INSTITUTE OF TECHNOLOGY, CHINA, 2003

A Thesis Submitted to the Graduate Faculty of the University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2008

INTEREST MANAGER MECHANISMS FOR DISTRIBUTED MULTI-AGENT-BASED SIMULATIONS USING AN OPTIMISTIC EXECUTIVE

by

TIANHAO HE

Major Professor: Maria Hybinette

Committee: Eileen T. Kraemer
Lakshmish Ramaswamy

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2008

DEDICATION

To my mother, I will miss you forever.

ACKNOWLEDGEMENTS

I would like to express great appreciation to my advisor, Dr. Maria Hybinette, for her endless patience, guidance and tremendous help through my graduate research and thesis writing. Her insightful suggestions, enthusiastic encouragement, and proverbs have made the completion of this research possible. I also wish to thank my committee members, Dr. Eileen Kraemer and Dr. Lakshmish Ramaswamy, for their valued input and guidance.

I would like to thank the other members of the SASSY research group who designed a stable and powerful framework, thank you, Yin Xiong and George Vulov. I also want to thank George Vulov separately as he provided great support and encouragement throughout my MS research. George implemented and designed the mechanisms for lazy cancellation and breaking ties for simultaneous events. Thank you, George.

I am deeply grateful to my girlfriend, Ting Yang, for her continued support and consistent encouragement.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Multi-agent simulation is becoming an increasingly popular tool in robotics (Balch, T. 1998; Gerkey et. al. 2003), social animal studies (Balch, T. et. al. 2005, Luke et. al. 2005; Minar et. al. 1996), and game theoretic research. However, these simulations suffer from serious performance and scalability limitations. The **S**calable **A**gent-based **S**imulation **Sy**stem (SASSY) framework developed by our research group aims to leverage advances in the field of Parallel Discrete Event Simulation (PDES). It provides an agent-based API but with the performance benefits of a PDES kernel. The SASSY architecture consists of two layers: a standard PDES kernel and middleware, which provides an agent-based API.

In SASSY, individual agents are programmed by the application developer using the standard agent-based sense-think-act paradigm. In order to support the agents in a PDES kernel, each agent is provided a proxy that "lives" in the PDES Simulation. The proxy serves to translate agent activities expressed in the agent based API appropriately into discrete events in the PDES kernel. An advantage to building the middleware atop a standard PDES kernel is that advances in the PDES paradigm can be transparently applied to speed up agent-based simulations.

SASSY's PDES kernel is based on the Time Warp synchronization algorithm (Hybinette et. al. 2006) and has been completed. In this thesis work we focus on the challenge of distributing and managing environmental state information so that agents moving about the simulated world can access it efficiently and consistently. Our interest management LPs (IMLPs) facilitate a publish/subscribe protocol between the agents themselves. This research centers on

comparing two methods: A peer-to-peer approach, and a method that uses intermediate LPs. Both approaches are fully distributed.

We describe the design, the application programmer interface and the implementation of the agent- based middleware extensions, along with a simple but powerful multi-agent simulation benchmark. The agent-based middleware aims to achieve parallel speedup by exploiting the *locality* usually encountered in agent-based simulations. Although in a multi-agent simulation all agents view and modify a shared environment, usually the actuating region of each agent is small compared to the overall size of the environment. Our interest management scheme can take advantage of this observation that agents typically have different sense and actuating radiuses and provide a performance advantage; we will discuss this in more detail in Chapter 4.

The agent-based paradigm and characteristics suggest that other advanced PDES techniques such as lazy cancellation can further improve performance. Lazy cancellation delays secondary rollbacks and caches the results of the original rollback in anticipation of reusing the results and thus avoiding re-computation (Gafni 1988). We discuss additions, such as Lazy Cancellation, and how it improves performance in the chapters below. But first we provide some background on PDES simulation systems and other related work.

CHAPTER 2

BACKGROUND

The work presented here is implemented and evaluated using and enhancing SASSY, a hybrid Agent-Based Modeling / Discrete Event Simulation (ABM/DES) System introduced in (Hybinette et al. 2006). For completeness we provide a brief overview here.

DES systems are fast and efficient because the systems they simulate are treated as if they proceed forward in discrete time steps – the intervening time is ignored. As compared to continuous time simulation, the discrete nature of time in DES systems enables a reduction in complexity because the requirement for synchronization is reduced. Furthermore, researchers have considered carefully how to gain speedup by distributing and parallelizing DES across multiple processing elements in **P**arallel **D**iscrete **E**vent **S**imulation (PDES) systems.

2.1 Parallel Discrete Event Simulation

Distributed parallel simulation provides at least two advantages. First, multiple processors can be used to reduce the execution time of the simulation. Second they may be required to support distributed personnel or resources (e.g., a combat simulator with multiple human participants at different locations). Distributed simulation also facilitates linking existing simulators developed for different platforms to model large systems.

In this work, a parallel simulation is composed of distinct components called logical processes or LPs. Each LP models some portion of the system under investigation. For example in an air traffic simulation an LP might represent each airport. The logical processes may be mapped to different processors. As in a sequential simulation, a change in system state is defined

by an event. The “scheduling” of an event is accomplished by sending a message from one LP that may request the destination LP to change its state or schedule additional events. For example processing a departure event may result in scheduling a new arrival event at another airport. Since events are scheduled by sending messages, “message” and “event” are used interchangeably in our discussion. To summarize: distinct components in the simulation are modeled by logical processes and the simulation progresses as LPs exchange time-stamped event messages that cause changes in the system state at discrete points in time.

A synchronization mechanism is used to ensure each LP processes events in time-stamp order. The two leading classes of synchronization protocols are *conservative* (Chandy and Misra 1981) and *optimistic* (Jefferson and Sowizral 1985) approaches. A conservative protocol enforces consistency by avoiding the possibility of an LP ever receiving an event from its past (as measured in simulated time). LPs *wait* to process events until reception of an out-of-order event is impossible. The optimistic protocol, in contrast, uses a detect-and-recover scheme. When an event is received in an LP's past, an LP recovers by rolling back previously processed events with later time-stamps than the one that was just received. While optimistic protocols are more complex and require more memory than conservative protocols, they offer greater concurrency by being less reliant on lookahead.

2.2 The Agent Based Paradigm

The standard PDES API for simulation developers is not well suited to agent-based applications because it does not offer the programming model these researchers expect. For example, multi-agent system (MAS) researchers expect to treat agents as objects that move around in an environment (like messages in DES, but with the ability to compute). In most PDES simulations LPs don’t move, they represent geographically static objects such as network routers,

airports, sectors in the airspace, intersections. So, in these simulators the objects that perform computing don't move. In contrast, in physical agent simulations the agents move around. In general, ABM researchers expect their agents to (Riley and Riley 2003; Balch 1998; Gerkey et al. 2003):

- **Use the Sense-Think-Act cycle** – agents sense their environment; consider what to do, then act. This is the predominant computational paradigm for agents; it stands in contrast to the message/event paradigm for PDES.
- **Compute** – Agents have computing capability and state; again, in contrast to messages in PDES, which provide no computing function.
- **Proliferate** – MAS simulations typically involve hundreds or thousands of agents.
- **Persist** – Agents are persistent members of the environment, in contrast to messages that exist only for a short periods.

For these reasons, a number of MAS and multi-robot systems researchers have devised their own simulation systems for their research. From a software engineering and ease of use point of view their simulators are well suited to the research tasks they pursue, but these simulators are not "high performance" in the same sense that state of the art PDES systems are. In fact, some agent based simulation systems face serious performance limitations. These limits prevent MAS researchers from investigating systems with thousand or millions of agents.

We feel the best solution is to provide middleware between a PDES kernel and agent-based API. This will enable MAS researchers to program using a model that is comfortable for them, while they leverage the high performance of an underlying PDES kernel.

2.3 SASSY Framework

Before we introduce our interest management approach we first briefly review our architecture. In the standard physical agent model, an agent senses its environment, considers what to do, then acts (see Figure 1. This is frequently referred to as the sense-think-act cycle (Riley and Riley 2003; Uhrmacher et al. 2000; Logan and Theodoropoulos 2001).

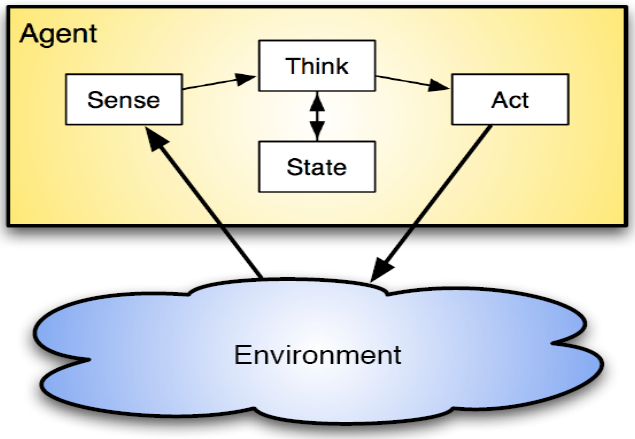

Figure 1: The Physical Agent Model.

Multi-agent simulators are typically configured as shown in Figure 2. The code for each agent connects to a process that maintains world state for the simulation. An Application Programmer's Interface (API) allows agents to query the simulator for sensor information and to send actuation commands to the simulator. The simulator updates the world state accordingly. The simulator checks for possible physical interactions that would prohibit a requested action. The simulator also moderates interactions between agents (such as communication).

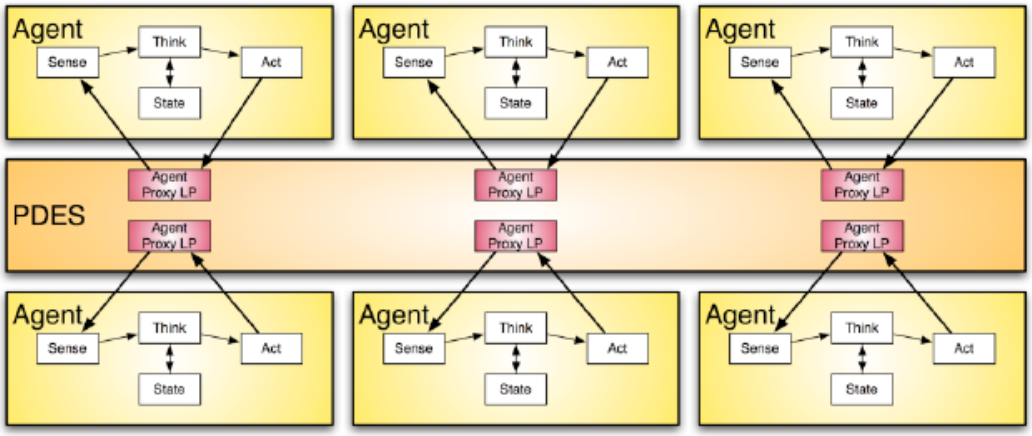

Figure 2: An Agent-Based Simulation.

The agents may be implemented in a number of ways. In TeamBots, for instance, agents are Java objects with "call back" methods the simulator calls to give them an opportunity to run (Balch 1998). In SPADES and Player/Stage the agents are separate processes that connect to a single-threaded simulation engine (Riley and Riley 2003; Gerkey et al. 2003).

SASSYs agent-based modeling API is implemented as middleware between a PDES kernel and the agent based model application code (see Figure 3). Each agent is provided a PDES proxy LP that serves to process and create messages. Figure 3 shows how this works for one agent.

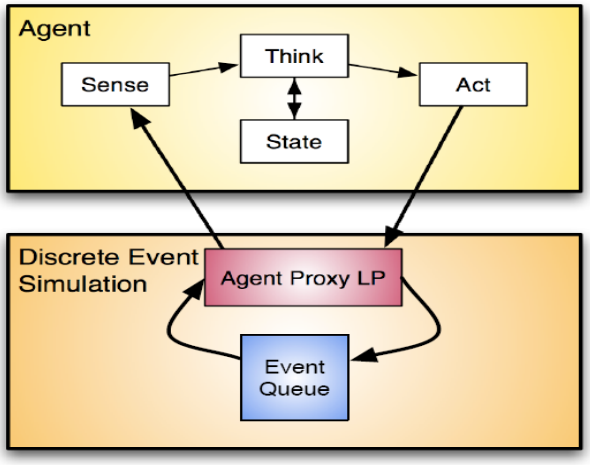

Figure 3: An LP in the PDES System Serves As a Proxy
for a Simulated Physical Agent.

Each agent proxy maintains a model of relevant objects in the environment near the corresponding agent it serves as a proxy for. When agents move or act in the world they generate an event that is sent to the other nearby agents so they can track the movements and state of others. The agent proxy LPs keep their state current for the agent they support.

As mentioned above, the Agent Proxy LP (APLP) keeps track of the world's state that is relevant to the agent it serves. In our approach, there is no central representation of world state. Instead, that agent's proxy LP maintains the world state relevant to each agent. As an agent's state changes, it notifies other agents using a state message reflection mechanism. Message reflection is accomplished by a distributed publish/subscribe mechanism implemented by a set of LPs arranged in a grid. These LPs are referred to as Interest Monitoring LPs (IMLPs). Each agent registers interest in (subscribes to) the activities that occur within specific cells. Agents that move within a specific cell periodically publish their state by sending a message to the relevant IMLP; then the IMLP reflects those messages to other interested agents.

CHAPTER 3

RELATED WORKS

The use of PDES for agent-based modeling is a relatively new idea that has been used to support research in soccer, biological systems and general purpose agent-based models (see for instance (Uhrmacher 2001), (Logan and Theodoropoulos 2001), and (Riley and Riley 2003)). However, we believe there are several aspects of our approach that contribute to a novel high-performance design. In particular, we use a "standard" PDES kernel, and we provide a "standard" agent-based model view. Because we use a standard PDES kernel we are able to easily leverage existing and future performance technologies such as optimistic protocols, distributed execution and advanced efficient Global Virtual Time calculations. Accordingly we make the simulation application developer's job easier -- she can more directly map her problem to the simulator without having to know the details of PDES.

Most research utilizing agent-based simulation centers on modeling autonomous agents (e.g., robots or automobiles) moving about a 2- or 3- dimensional environment. These agents rely on a *sense-think-act* cycle where they sense information about the local environment, think about the information in the context of their own behavioral state, and then act in the environment. From the point of view of a simulation kernel, these activities correspond to reading state, processing it, and writing new state. A key challenge concerns maintaining a consistent environmental state that agents can sense (read) and act upon (write).

Logan and Theodoropoulos provide a comprehensive and readable description of this problem in (Logan and Theodoropoulos 2001). Their solution centers on Environmental LPs (ELPs) in which the environmental state is managed and distributed. However their experimental

results only include **1** central environmental LP with **64** agents (Lees et. al. 2007). Our approach is somewhat different, in that state is maintained by the agents and objects in the environment directly to provide an intuitive API for the ABM programmer. Also, our experiments include runs with 3,000 or more interacting agents and 400 IMLPs.

Our interest management LPs (IMLPs) facilitate a publish-subscribe protocol between the agents themselves. This is similar to High Level Architecture (HLA) (Dahman et. al. 1997) interest manager approaches that use conservative clocks (e.g. Tacic and Fujimoto's work reported in (Tacic and Fujimoto 1998) and Wang, Turner and Wang's work in (Wang et. al. 2003)). Tacic and Fujimoto's work focuses on reducing network traffic in a simulation using a conservative protocol (HLA) while Wang, Turner and Wang describes how to integrate agents using different interest management schemes into an HLA-based distributed simulation. In contrast, our approach supports optimistic simulation (Jefferson and Sowizral 1985; Fujimoto 1990), and performance improvement centers on reducing the number of rollbacks.

Logan and Theodoropoulos also propose a related approach to our interest management for an optimistic simulator in (Logan and Theodoropoulos 2001). In their approach world state is maintained in “environmental LPs” (similar to our IMLPs), however the ELPs kept state centralized. In our approach the agents track world state themselves; there is no centralized representation of any agent’s state.

Also of interest is the fact they address the degree coupling and skew in optimistic simulators. In the SASSY simulation framework, evenly allocating agent LPs and IMLPs to participating PEs can minimize the skew. In addition, in SASSY one can vary the number of IMLPs to achieve different degree of coupling (e.g., un-coupled, coupled or fully coupled). As the number of IMLP increases, the agents far from each other become uncoupled with each other.

This enables SASSY to be both efficient and flexible. In this work we do not assess how skew impacts the performance of our approach, but we intend to address this in future work.

In SPADES and Player/Stage the agents are distributed and run as separate processes that connect to a single threaded simulation engine (Riley and Riley 2003; Gerkey et. al. 2003). Because these other simulation systems utilize a central resource (the simulation server) they are not able to scale well on a distributed computer platform. However, SASSY uses a standard PDES kernel and is able to leverage the corresponding benefits. Our SASSY kernel supports the optimistic synchronization paradigm which is one of the standard synchronization protocols used in PDES (Jefferson and Sowizral 1985; Fujimoto 1990). Performance improvements for Optimistic PDES systems center on reducing the cost of rollbacks and scalability on distributed computing platforms. Among the various performance enhancements available to PDES systems, SASSY leverages distribution of multiple processors, function caching (Xiong et. al. 2008) and lazy cancellation (Vulov et. al. 2008) and re-evaluation.

CHAPTER 4

INTEREST MANAGEMENT APPROACHES

As described in Chapter 2 each agent is represented by a proxy logical process (LP), which maintains an up-to-date version of the environment currently visible to the agent. For efficiency and scalability the SASSY middleware leverages the domain decomposition method by dividing the environment in a number of 3D rectangular regions, each of which is managed by an Interest Manager Logical Process (IMLP). The IMLP implements a subscribe/publish system for the proxy LPs to ensure that all agents have a consistent view of the shared environment.

It should be noted that the distribution of the global state amongst IMLPs is completely transparent to the multi-agent application. Indeed, the granularity of the distribution of the global environment can be controlled with a parameter loaded at runtime. Developers of agent-based simulations can adjust the size of the IMLP regions to suit the needs of the particular application. Ideally, the size of a region controlled by an IMLP should be roughly equal to the area that an agent can directly modify.

Proxy LPs can send five basic types of messages to an IMLP:

1. SUBSCRIBE
2. UNSUBSCRIBE
3. ENTER
4. LEAVE
5. UPDATE

A SUBSCRIBE message lets the proxy LP monitor updates in a certain region without committing any changes to the region. The IMLP sends a description of the current region state back to the new subscriber. The SUBSCRIBE message is not relayed to other agents currently in the region, since the observing agent cannot influence their actions. Correspondingly, UNSUBSCRIBE message removes an observing agent and is not relayed.

An ENTER message notifies the IMLP that the agent is going to be modifying the managed region. A modification of the environment can be something as simple as the agent moving its body into the region. An ENTER message is relayed to all other agents subscribed to the region, since the new entry could potentially influence their behavior. Similarly, a LEAVE message indicates that the proxy LP will not be committing any more changes to the region and is also relayed to all other subscribers.

Finally, the UPDATE message is used by proxy LPs to notify other agents' proxy LPs of changes in the observable environment. An UPDATE message is relayed by the IMLP to all subscribers. The IMLP also processes the message, maintaining an up-to-date local copy of the observable environment in the region.

4.1 Event-driven Implementation of the Agent Interface

Multi-agent simulations typically progress by evaluating a sense-think-act cycle for every agent in the simulation. Typically the thinking time varies from 10ms to 1000ms (Balch 2008: Riley and Riley 2003; Lees et. al. 2007). Agent-based robot simulators usually assume a fixed time period between the incoming sense events; for example, a 33 msec time step could be imposed by the frequency of the video sensors (TeamBots (Balch 1998) and Player/Stage (Gerkey et. al. 2003)). The SASSY middleware relates the PDES virtual time to the agent's simulated time through a constant multiplier, Δ. Hence, in a SASSY agent-based simulation,

time flows in discrete intervals of time Δ. Each agent can specify how many intervals pass between the invocations of its sense-think-act cycle. For example, consider a simulation with two robot types: type A senses every 50 msec and the type B senses every 10 msec. The SASSY middleware would be configured with Δ = 10 msec. Robot type A would receive a sense callback every five time intervals, while type B would receive a callback every time interval.

The agent's sense callbacks are implemented by its proxy LP. In addition to sending messages destined for the IMLP (described previously), each proxy LP schedules a SENSE message to itself. Each time a proxy LP processes a SENSE event, it schedules its next SENSE event. The simulation time advancement of the SENSE event depends on the agent's processing rate. If an agent's time progresses in 20 msec intervals and Δ = 10 msec, then the timestamps of its SENSE events would increase by two each time. Note that due to the event-driven nature of the PDES architecture, there is no performance punishment for using a lower discrete time step than an agent's processing time. There are no "slots" wasted by having an agent process once every ten intervals rather than every interval. Taking a SASSY agent simulation and halving Δ would result in doubling of the timestamps of all the underlying PDES messages; such a simulation would perform no more computations than the original.

4.2 The Relay Approach

We are exploring two approaches to interest management. In this section, we describe the Relay Implementation first introduced (Hybinette, Kraemer et al. 2006). Consider the scenario illustrated in Figure 4. In this case, we are concerned with agents A, B, C, and D. A and B will register interest in the light colored cell (for convenience we will refer to it as IMLPj). C and D will move through the J cell and post state messages to IMLPj. In a real simulation, all four agents would subscribe to multiple IMLPs and they would also all post state information to the

IMLP cell they travel within. In this example, we focus on the messages related to A and B acting as subscribers and C and D acting as publishers.

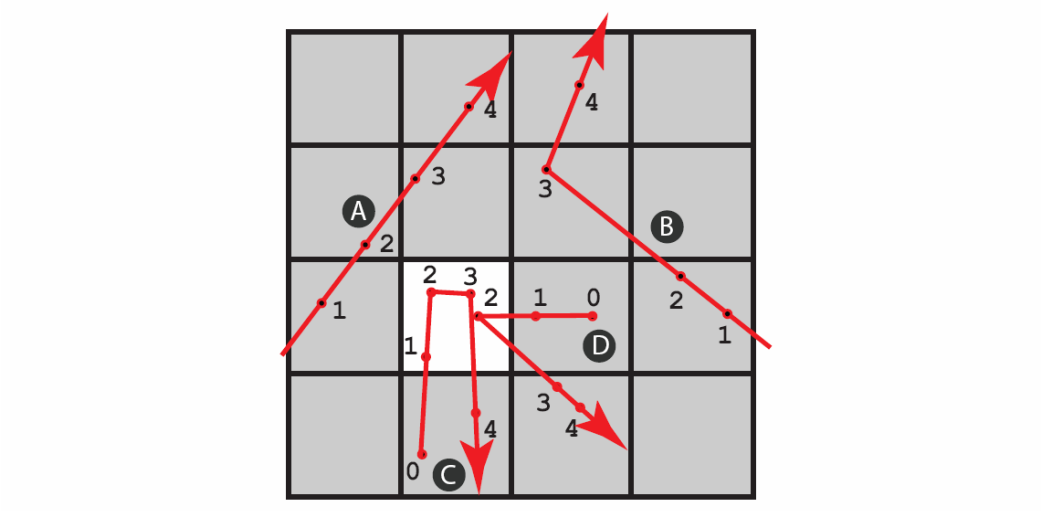

Figure 4: Four Agents: A, B, C and D That Roam About a 2 Dimensional Space. The Light Colored Region Is an Interest Region Maintained by IMLPj. Positions of the Agents and Their Directions Are Denoted by Dots and Arrows (the Numbers Refer to Instants in Simulated Time)

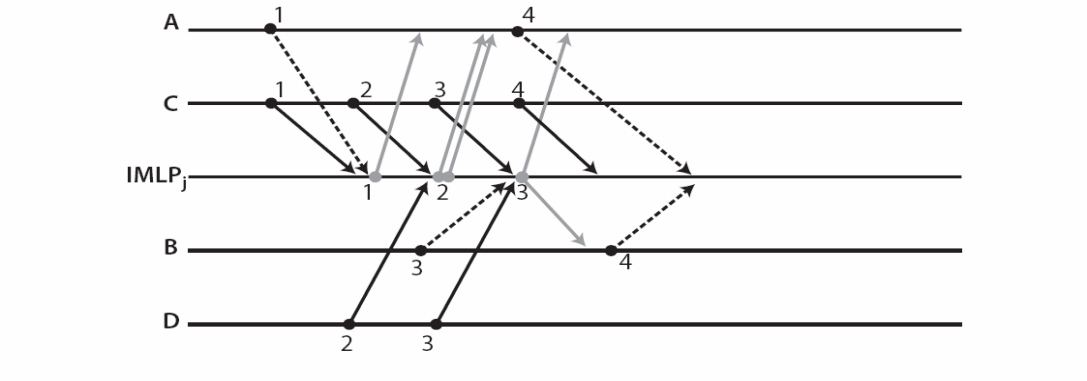

Figure 5: Event Message Timeline for Agent LPs A, B, C and D and IMLPj

In Figure 5, we show an example timeline of event messages sent to and from IMLPj. In this timeline, all four agents are roughly synchronized. Events occur as follows: Agent A subscribes to information from IMLPj at time 1 and unsubscribes at time 4 (all times are given in

simulation time). Agent B subscribes at time 3 and unsubscribes at time 4. Agent C enters cell J at time 1, it sends an enter message at time 1, then states messages at time 2 and 3. Agent C leaves cell J at time 4 and sends a corresponding leave message at that time. IMLPj receives the state messages from C and "reflects" them to A and B at the appropriate times. Agent D enters cell J at time 2 and leaves at time 3; it sends appropriate enter and leave messages at those times. IMLPj reflects the time 2 state message from D to agent A at time 2. Note that Agent B does not have to be notified of D's activities, because it was not interested in IMLPj at time 2. In this scenario no rollbacks were necessary. IMLPs maintain a record of state messages and subscription windows back to Global Virtual Time (GVT). If a state message arrives within the subscription window of a particular agent, the IMLP will reflect that message to the interested agent. Also, if later in time, an agent registers interest in events during a time window in the past, the IMLP will reflect those old messages from that time to the agent. Now consider the timeline in Figure 6. Events move forward in a manner similar to the earlier example, except that agent D is somewhat behind the other agents. In real time, agent D arrives in cell J after C has come and gone and after A and B have unsubscribed to information about cell J. D's message at time 2 is reflected to agent A. A is forced to rollback, because it had already advanced in simulated time to time 4. Note that agent B does not have to rollback, because it is not affected by D's activities. B does not have to rollback because it is not affected by D's activities, either.

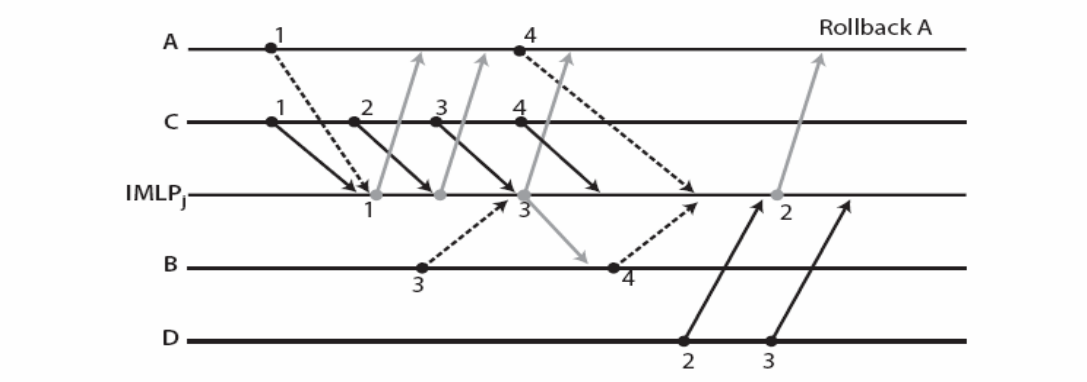

Figure 6: Time Lines of Agents (Agent LPs): A, B, C and D
and Interest Manager LP That Corresponds to a Cell in the Grid
This Example Shows That Agent A Needs to Roll Back

4.3 The Peer to Peer Approach

One potential failing of the relay implementation is the overhead for each message having to be sent twice (from one agent proxy to the IMLP, then to another agent proxy). In the peer-to-peer approach, the interest manager serves only to hook up agents that are interested in monitoring a particular area with those agents from the reduced message traffic. In this section, we describe a peer to peer communication approach that avoids relaying messages. Consider the scenario illustrated in 7s that shows a time line of event messages sent to and from IMLPj and agents A and C. To simplify the discussion, we will not discuss interactions involving agents B and D.

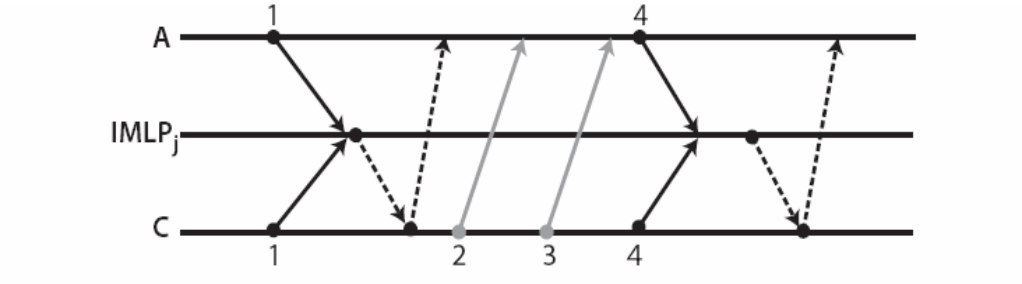

Figure 7: Event Message Timeline for Agent LPs A, C and
IMLPj using the Peer to Peer Mechanism

At time 1, agent A subscribes to cell J that is managed by IMLPj and, roughly at the same time, Agent C enters J. C notifies IMLPj that it is a publisher. IMLPj then reflects an initial subscription message to C. C adds A to its neighbor list. Subsequently, C sends an update message directly to A without involving or relaying the message via IMLPj. Agent A receives update messages from C at time 2 and 3. Agent A unsubscribes to cell J and notifies IMLPj at time 4. The unsubscribed message of A is reflected to C. C sends an acknowledgment message to A. A and B clears and updates their neighbors' lists and terminates the connection between them. In this approach, message traffic is reduced as only one message and is sent for each update action (instead of 2 in the relay approach). This may be especially beneficial when there is remote communication; however, if the message traffic is mainly local, the performance benefit may not be as significant.

4.4 Relay Approach with Lazy Cancellation

As discussed before, Peer-to-Peer approach can reduce communication overhead for update messages. However, the relay mechanism can be improved to enable IMLPs to act as a buffer between agents at different simulation times. As an example, consider a fast-processing agent, F, which passes through a region and sends UPDATE messages. Some physical time later, a slower-processing agent, S, subscribes to the region. This is depicted in Figure 8. The 'region' is light and agent F is in the region at the time steps 2, 3, 4 and 5 are at a wall clock time before S. S enters the region at simulated time 2.

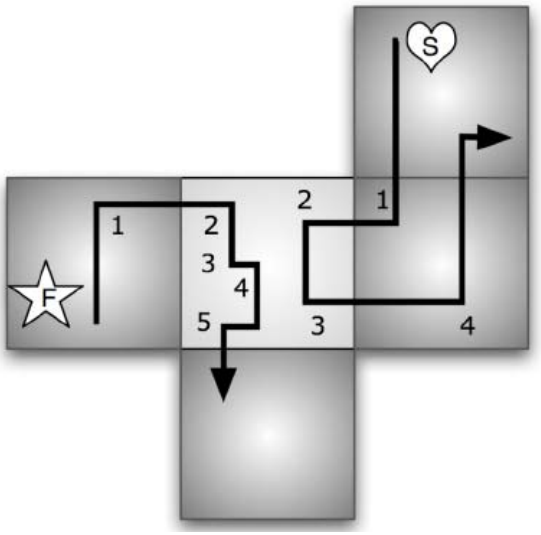

Figure 8: Example of an Agent F Entering a Light Color Region
and an Agent S Entering the Same Region.

In a peer-to-peer communication schema, S's SUBSCRIBE message would roll back F's time and force F to re-process its movements through the region (note a rollback of F may include F canceling messages it sent to other subscribers). This is illustrated in the lower image in Figure 9; here S's message at simulated time stamp 2 rolls back F's messages with later time stamps.

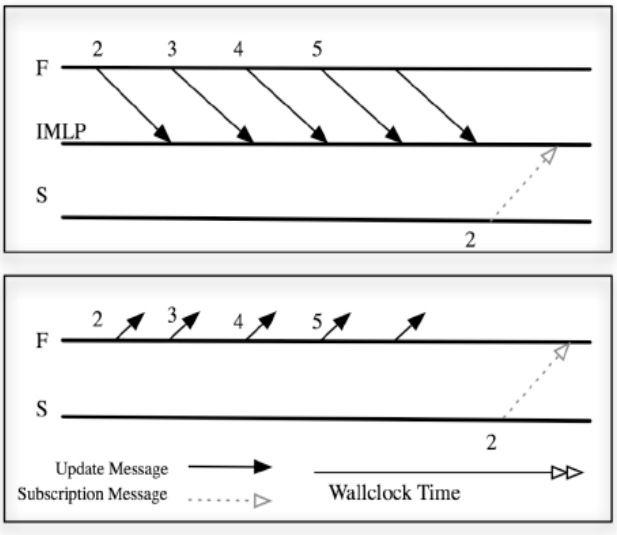

Figure 9: Message Progression as a Slow-Processing Agent
Enters a Region Where Another Agent has Passed

On the other hand, if S subscribes to the IMLP, the IMLP should be able to replay F's UPDATE messages to S without rolling back F (and, in turn, possibly canceling message updates to other subscribers). This scenario is illustrated in the top image of Figure 9. Here the IMLP forwards the buffered messages with time stamps later than S's subscription message. Note that this avoids rolling back and canceling messages to other subscribers.

Though we would like the IMLP to act as a buffer between agents of different speeds, recall that the middleware runs on a standard optimistic PDES kernel. Thus, a rollback of the IMLP could force it to transmit anti-messages, causing all agents that have previously passed through the region to be rolled back. In other words, a slow-processing agent could force all faster agents to "come back" to a region just by observing it. To achieve the desired property that slow agents can observe regions without affecting the rate of computation there, we implement a well-established optimistic technique: lazy cancellation.

When a logical process is rolled back with lazy cancellation, its anti-messages are not sent out immediately (Gafni 1988). Rather, the logical process is left to coast back to the pre-rollback time. If during the coasting phase it sends the same messages as before, there was no need to send the anti-messages in the first place. Anti-messages are sent out only if the LP does not regenerate the same messages as before.

When an IMLP receives a SUBSCRIBE request from the past, it rolls back to that time. When coasting forward, it re-sends (or reflects) all UPDATE messages. However, a subscriber cannot modify the state of the environment, so all reflected updates would be the same as the ones sent before the rollback. Thus, if lazy cancellation were applied to rollbacks of IMLPs, late agents subscribing to a region would not cause rollbacks of other agents in the region. This can be a big performance boost since rolling back an IMLP is relatively inexpensive, while rolling

back an agent can cause it to re-compute think-sense-act cycles taking 10ms to 1000ms each (Balch 2008: Riley and Riley 2003; Lees et. al. 2007).

CHAPTER 5

EXPERIMENTAL AGENT-BASED SIMULATOR

5.1 Bouncing Ball Agent Benchmark

The multi-agent simulation used for performance testing consists of a number of bouncing balls. Each ball is an independent agent with a body of a certain radius (which is configurable). The ball's actuating region consists of the immediate space that its body occupies, because the only change that a ball can make to the environment is moving throughout it. A ball's sensing region is of configurable radius, but it must be at least as large as its body. Each ball also has a color attribute, which it updates based on the number of other balls it detects in its sensing region. When two balls detect a collision, they bounce off of each other by exchanging their velocities.

As described, the Ball agents are simply reactive; there is very little computation involved in deciding what to do at each time step. We would like to evaluate the performance of SASSY with deliberative agents, for instance agents running A* at every time step as in (Lees et. al. 2007). To simulate similar processes, every Ball agent also incorporates a random deliberative delay, which varies in a flat distribution between the minimum delay and the maximum delay. An important property of the simulation model chosen is that an agent's behavior (color) is influenced by its observations in its entire sensing region; computation can be rolled back by any update in the sensing region.

Our benchmark is similar to the 'bouncing ball' benchmark reported in SPADES (Riley and Riley 2003), where the authors report performance of 2-26 agents/balls. Balls in our SASSY environment are basically brainless bodies of more sophisticated agents such as ants and bees.

The benchmark includes an optional graphical display and the display is depicted in Figure 10. Here, the simulator is running with 10*10 IMLPs and 20 ball agents. A single small grey square region denotes an IMLP. The ball agents are evenly distributed over the environment with an initial velocity sampled from a uniform distribution. As the simulation progresses, the balls may collide and exchange inertia. The small number in the center of the balls denote the number of neighbors sensed and it is also reflected in the color of the ball, the larger the number of neighbors the more intense the color.

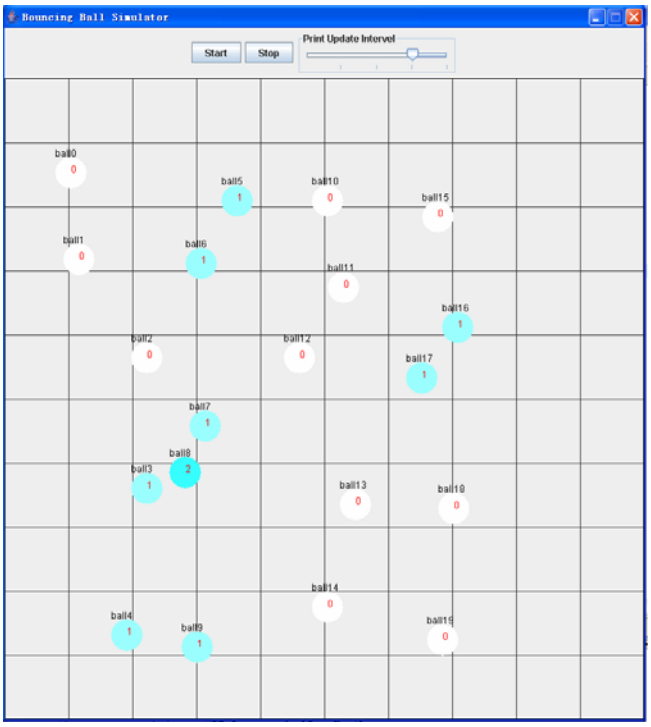

Figure 10: Bouncing Ball simulator screenshot
(in this case the sensor region is equal to the actuating region)

5.2 System Parameters:

Table 1 lists all configurable parameters in the system.

Parameters	Comments
NUM_BALL	Number of the ball agents
NUM_IMLP	Number of interest manager LPs
NUM_PE	Number of worker processor elements(machine)
BALL_RADIUS	Radius of a ball agent
NUM_MOVE	Total number of the ball movement during the simulation
IM_TYPE	IM communication type : direct/p2p
GUI_USED	Whether or not the GUI been used
ENVIRONMENT_X	The length of the environment x dimension
ENVIRONMENT_Y	The length of the environment y dimension
VELOCITY_X	The ball's x dimension velocity(average)
VELOCITY_Y	The ball's y dimension velocity(average)
DELIBRATION_TIME	The ball's thinking time(average)
LAZY_CANCELLATION	Whether or not the lazy cancellation mechanism is used
SYMMETRIC	Whether or not the balls have symmetric sensing region. If this set to false, the odd number balls have larger sensing region than even number balls do.

Table 1: Bouncing Ball simulator parameters

CHAPTER 6

EXPERIMENT RESULT AND DISCUSSION

6.1 Distributed Performance

In our first set of performance tests, we evaluate the scalability of SASSY, in terms of number of agents and the number of machines. The simulation environment ran with dimensions 1000x1000x1, distributed among 100 IMLPs. Each ball agent had a radius of 10, with equal sensing and actuating regions. The deliberation time for an agent was on average 300 msec, varying uniformly from 250 to 350 msec (note that this range of thinking time is typical for agents in ABS, see Chapter 2).

Furthermore, we compared the performance of SASSY to the performance of a hand-coded time-stepped serial simulator programmed specifically for our agent setup. All tests were executed on Sun workstations, networked together with a gigabit Ethernet. Each workstation had a dual-core AMD Opteron running at 2.6 GHz with 4 GB of RAM.

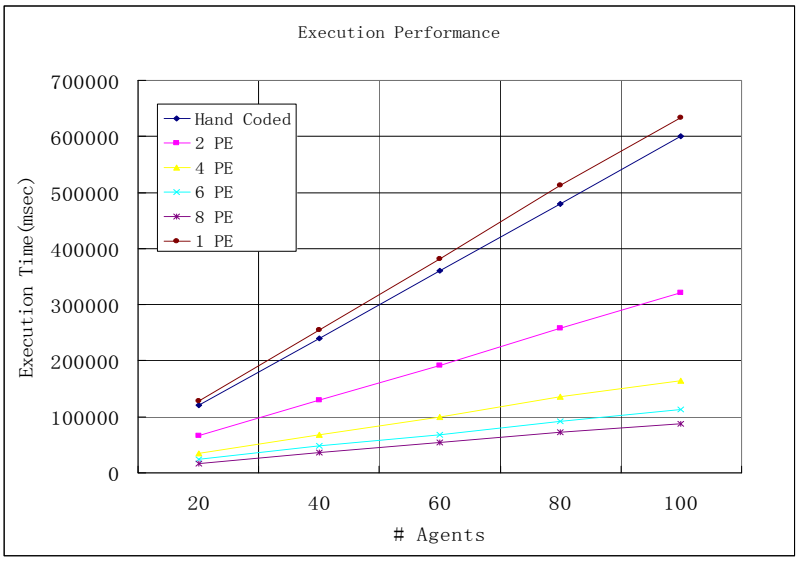

Figure 11: Execution Time with Growing Number of Agents

Figure 11 illustrates the performance of SASSY as the number of agents increases. We ran 6 sets, on 1, 2, 4, 6, 8 PEs. When executing on one PE the hand-coded simulator outperforms SASSY due to the overhead of the SASSY middleware. Fortunately, the overhead is slight; and induces a small price to pay for the ability to seamlessly distribute the simulation across multiple machines. SASSY can be run on 2, 4, 6, and 8 machines simply by changing a configuration parameter and starting the specified number of clients. It offers a substantial speedup over the serial simulation without requiring additional effort from the multi-agent modeler.

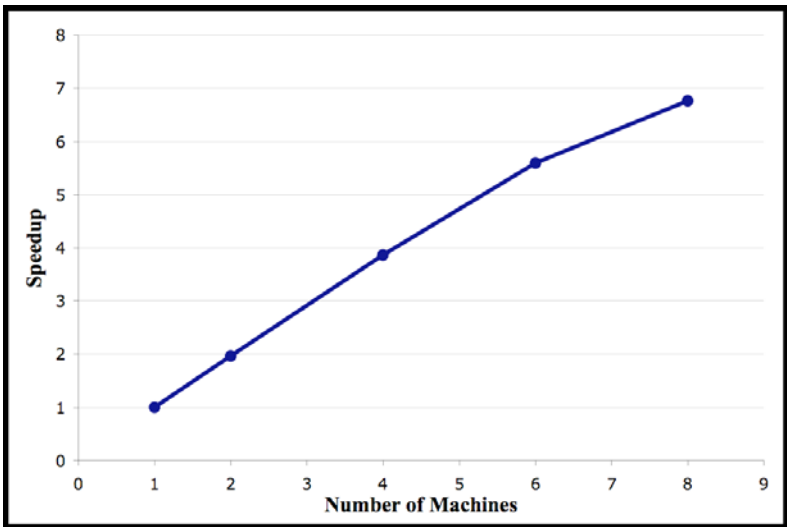

Figure 12: Distributed Execution Speedup
(100 Agents, 300 msec deliberation)

Figure 12 shows the speedup of SASSY when running the simulation with 100 agents on up to eight machines. Communication overhead is the major factor that prevents SASSY from reaching a theoretical maximum speedup of 8x. One might argue that the agents' high (300 msec) deliberation times and the low number of agents are masking a rather significant communication overhead. For this reason, we ran the same simulation described above with 3000 agents, each of which had a very short deliberation time.

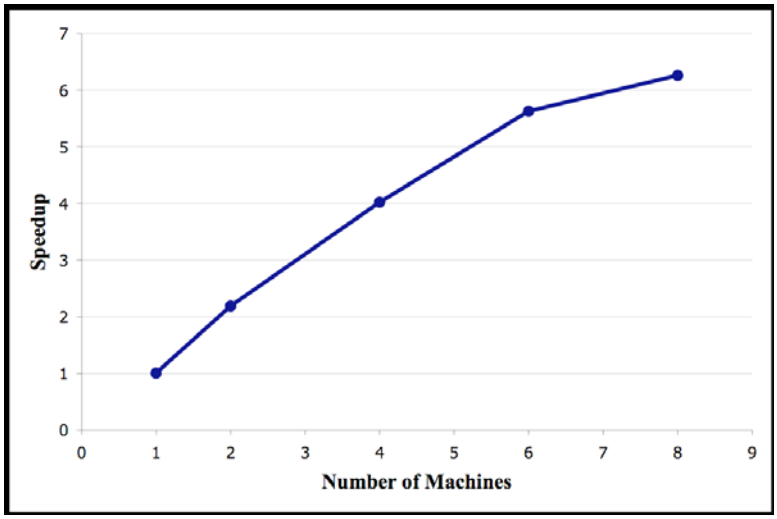

Figure 13: Distributed Execution Speedup
(3000 Agents, 5 msec deliberation)

The speedup gain in Figure 13 is clearly less linear than the speedup gain in Figure 12; nevertheless, SASSY still achieves a 6.2x speedup when executing with 8 PEs. Increasing the number of agents by 30-fold did not significantly slow down the simulation executive and decreasing the agents' thinking times did not reveal any unusual communication overhead.

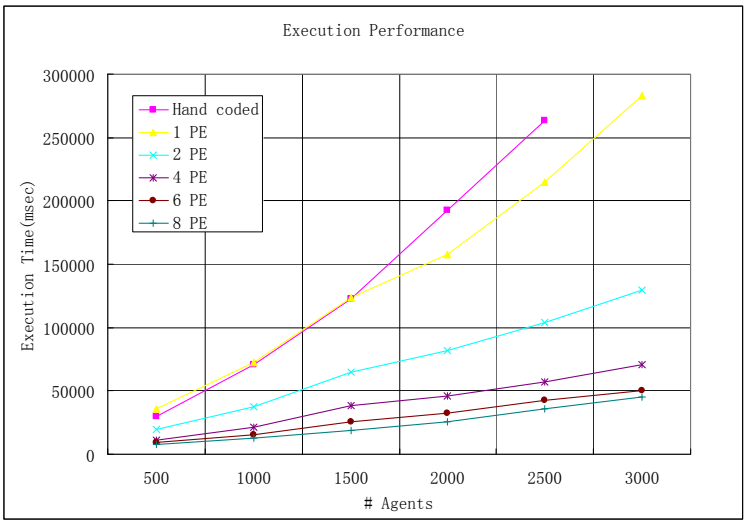

Figure 14: Scalability Performance for Large Number of Agents

Besides the simulation execution time, resource limitation is another issue that prevents a centralized simulator to scale to a large number of agents. Figure 14 shows the scalability performance for a large number of agents. For the Hand-coded version, since no interest management schema is used, state updates are broadcast to all other agents. Further, in a serial simulation all agents execute in the same memory space. The resource usage feature for such a traditional centralized agent simulator makes the communication and memory become a bottleneck. As shown in the figure, the hand-coded version do not scale well as the number of agents becomes large. When the number of agents increases to 2500, the hand-coded version is apparently slower than 1 PE version. When we try to simulate 3000 agents, the hand-coded

version exhausts physical memory and cannot complete the simulation. Thus, we can conclude that the maximum number of ball agents that a traditional centralized simulator can handle is less than 3000, and the situation will worsen when we simulate complex agents. The 1 PE version, on the other hand, scales better than the hand-coded version by taking the advantage of the SASSY interest management mechanism. Update states are multicast only to interested agents so that a great amount of memory space for state saving is saved. The simulation can execute on relative low memory usage, and thus, achieves better scalability performance.

6.2 Performance with Agents with Varying Sensing Distances

In our discussion of relayed versus direct communication, we noted that relayed communication with lazy cancellation can offer substantial performance benefits when agents observe a region without modifying it. This situation is quite common; for example, a robot can have a video camera with a wide view but rather short actuators.

To test our hypothesis, we executed a series of simulations in which all factors were kept constant except for the sensing distance of the ball agents. The environment size was 1000x1000x1, managed by 400 IMLPs each covering a 50x50x1 region. The 100 agents had an average thinking time of 50 msec, varying uniformly from 0 msec to 100 msec. All tests were in distributed mode using four machines.

In the first simulation setup, all UPDATE messages were transferred directly from agent-to-agent without being relayed through an IMLP. In the second setup, an IMLP was used to relay environment updates, but no lazy cancellation was used. In the final configuration, IMLPs were used to relay UPDATE messages and lazy cancellation was enabled for IMLPs.

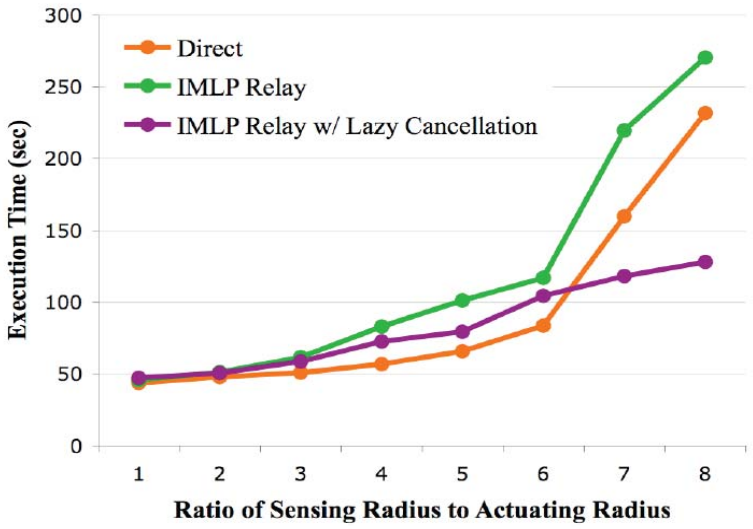

Figure 15: Execution Performance with Increasing Agent Sensing Distance

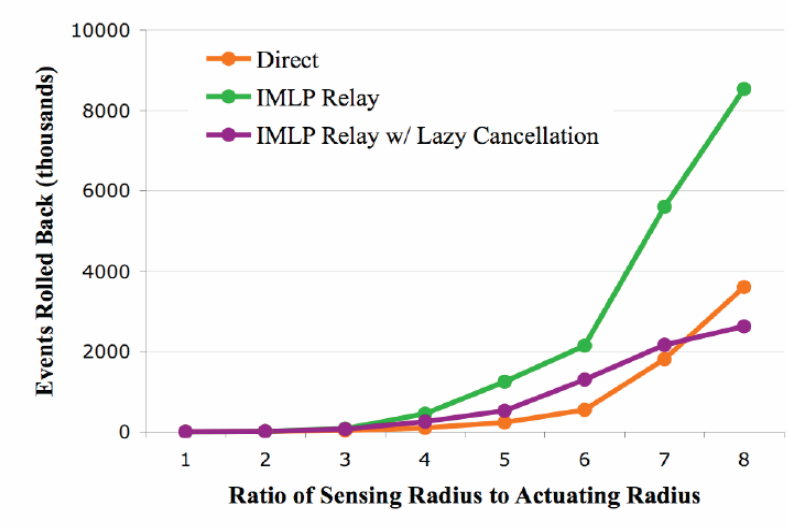

Figure 16: Number of Events Rolled Back with Increasing Agent Sensing Distance

Figures 15 and 16 show the results from the same executions of the simulation while varying the sensor to actuator ratio. Figure 16 shows number of rollbacks and Figure 15 shows execution time. When the agents have a relatively short sensing region relative to their actuating region, direct communication of updates between agent LPs is the fastest. The lower

performance of relay communication is due to communication overhead; there are roughly twice as many messages sent when in relay mode. Even so, relay communication with lazy cancellation is not significantly slower than direct communication.

As agents' sensing distances increase, performance of direct communication quickly degrades for the reasons described in section 3.2. Without lazy cancellation, agents can slow down computation in a region just by observing it. Note that the area that agents observe grows quadratically with their sensing radius; correspondingly, the performance of the communication methods without lazy cancellation worsens quadratically. With IMLPs and lazy cancellation, agents can only affect the computation in their actuating region; therefore, performance remains relatively good as the agents' sensing radii are increased.

It is interesting to point out that when the sensing/actuating ratio is 7, direct communication actually rolls back fewer events than relaying with lazy cancellation, but overall performance is worse. The discrepancy occurs since lazy cancellation prevents IMLP rollbacks from propagating to agent rollbacks while all direct communication rollbacks are agent rollbacks. Rolling back an agent's SENSE event and re-processing it can cost 50 ms or more (the agent's deliberation time), while rolling back and re-processing IMLP messages is very quick.

6.3 Best IMLP Region Size

While small regions can distribute memory overhead, it induces communication overhead. Our next set of empirically tests for IMLP size relative to ball size while minimizing communication overhead. The environment size was 1000x1000x1 and the ball radius was 10. The number of IMLP varied from 169 to 1000 and the corresponding IIMLP size was 0.5, 1, 2, 3, 4 times the ball size. Number of agents was 100 and they had an average thinking time of 50

msec, thinking time varied uniformly between 0 msec to 100 msec. All experiments ran on four machines.

Figure 17 shows the number of rollbacks while varying the IMLP cell size (cell/agent ratio). As expected, the smaller the IMPL/agent ratio, the fewer the number rollbacks. The reason is that while the IMLP cell becomes smaller, the agent sensing radius becomes smaller as well, and agents far from each other can process ahead without rolling back the slower one. However, when the size of IMLP cell is smaller than the agent cell, the number of rollbacks increases. This is because as the number of IMLP increases, the more IMLPs roll back, and thus the overall number of rollbacks increases.

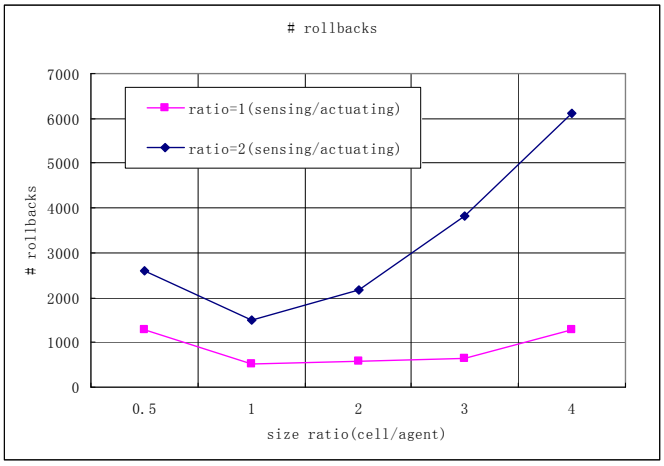

Figure 17: Number of Rollbacks for Different IMLP Region Size

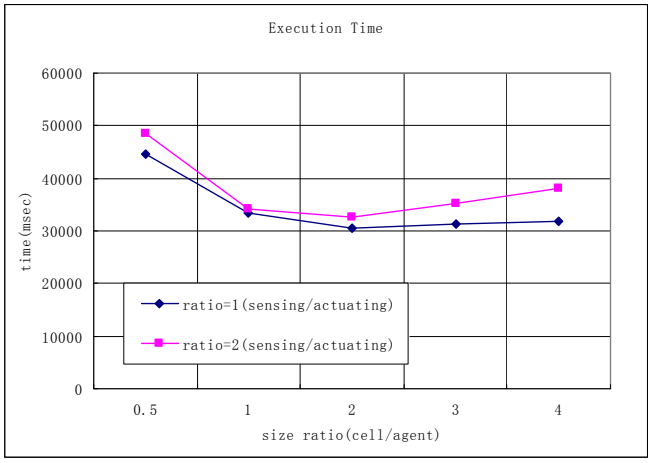

Figure 18: Simulation Time for Different IMLP Region Size

Figure 18 shows the execution time for different sizes of IMLP cells. Although the number of rollbacks is the least when IMLPs and agents have similar size, it is more efficient when IMLP is slightly larger than the agent. The reason is that communication overhead outweighed the cost of rollbacks. In conclusion, the tradeoff between the communication overhead and the cost of rollbacks is the main consideration while choosing the right size of IMLP.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The SASSY architecture has three distinct components: the PDES kernel, the agent-based middleware, and the agent testing application. We have demonstrated the feasibility and scalability of the SASSY design in several ways; however, there are a number of extensions we would like to examine in the future. The SASSY PDES kernel can be modified as previously described to apply lazy cancellation on a per message basis. The agent-based middleware should then attach lazy cancellation hints to message types in such a way to maximize performance. The SASSY PDES kernel should also continue to incorporate techniques developed by the optimistic PDES community to speed up the performance of agent-based simulations. One potential candidate is using infrequent state saving, which would lower a simulation's memory usage by increasing the length of its rollbacks.

For efficiency, we must consider serialization of LP-to-LP messages and the way those messages are transported across the network. The SASSY kernel API allows LPs to send Java objects to each other, which must be serialized somehow for network transfer. Currently, Java's built-in serialization is used, but perhaps a more restrictive custom serialization scheme will offer higher performance. For message transport, SASSY currently uses Java's Remote Method Invocation (RMI) mechanism. In the future, we plan to replace this with our custom protocol implemented directly over TCP, achieving higher message throughout, lower latency, and lower CPU usage. The SASSY agent-based middleware can be made both more flexible and faster. As mentioned before, some fairness can be introduced in the simulation by rotating the order in which agents move. Also, in the current work, the regions assigned to IMLPs are sized

uniformly. However, it may be useful for the regions to vary in size, perhaps because certain regions may be expected to contain only a few agents (See Figure 19). The agent-based modeling API provided by the middleware can also be improved by accommodating additional use cases.

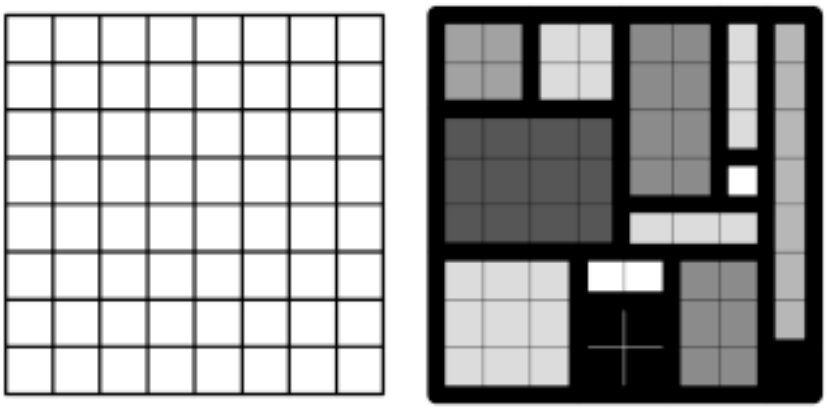

Figure 19: Current IMLP Regions vs.More Flexible IMLP Regions

For the testing application, in future work, we plan to develop more comprehensive and realistic agent-based test cases. One agent-based simulation will actually contain deliberative agents, perhaps ones running an A* search at every step. Another direction would be to explore the performance of simulations with purely reactive agents, such as ones found in some social animal studies.

REFERENCES:

Aaron, S. and P. Riccardo 1996. "SIM AGENT: A toolkit for exploring agent designs." Intelligent Agents Vol II (ATAL-95): 392--407.

Balch, T. 1998. Behavioral diversity in learning robot teams. College of Computing. Atlanta, Georgia Institute of Technology. Ph.D.

Balch, T. 2005. How multirobot systems research will accelerate our understanding of social animal behavior. In Proceedings IEEE. Volume 94; Numb 7: 1445-1463.

Balch, T. 2008. Personal Communication, College of Computing, Georgia Institute of Technology.

Dahman, J. S., R. Fujimoto and R. M. Weatherly. 1997. The department of defense High Level Architecture. In Proceedings of the 1997 Winter Simulation Conference (WSC-1997), 142-149.

Fujimoto, R. M. 1990. "Parallel Discrete Event Simulation." Communications of the ACM 33: 30-53.

Fujimoto, R. M. 2000. Parallel and distributed Simulation Systems. 1 st. John Wiley & Sons.

Gafni, A. 1988. Rollback mechanism for optimistic distributed simulation systems. SCS Multiconference on Distributed Simulation.

Gerkey, B. P., R. T. Vaughan, and A. Howard. 2003. The Player/Stage Project: tools for multi-robot and distributed sensor systems. 11 th Int. Conf. on Advanced Robotics (ICAR 2003) Coimbra, Portugal.

Hybinette, M., E. Kraemer, Y. Xiong, G. Matthews and J. Ahmed. 2006. SASSY: A design for a scalable agent-based simulation system using a distributed discrete event infrastructure. In Proceedings of the 38th Winter Simulation Conference (WSC-2006), 926- 933.

He, T. and M. Hybinette, 2008. A comparison of interest manager mechanisms for agent-based simulations using a TimeWarp executive. 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS-2008)

Jefferson, D.R., H. Sowizral. 1985. Fast concurrent simulation using the Time Warp mechanism. In Distributed Simulation 1985, Volume 15 of Simulation Council Proceedings, 63-69. Society for Computer Simulation (SCS).

K.M. Chandy and J. Misra, 1981"Asynchronous Distributed Simulation via a Sequence of Parallel Computations", Communications, ACM 24, pp. 198-205, Nov 1981

Les, G. and K. Kelvin 2002. MACE3J: fast flexible distributed simulation of large, large-grain multi-agent systems. Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2. Bologna, Italy, ACM.

Lees, M., B. Logan, and G. Theodoropoulos 2007. Distributed simulation of agent-based systems with HLA. ACM Transactions on Modeling and Computer Simulation. Volume 17:11.

Lees, M., B. Logan, T. Oguara, and G. Theodoropoulos 2004. "HLA_AGENT: distributed simulation of agent-based systems with HLA." Proceedings of the International Conference on Computational Science (ICCS'04) (pp. 907-915).

Logan, B., Theodoropoulos, G. 2001. The distributed simulation of agent-based systems. IEEE Proceedings Journal, Special Issue on Agent-Oriented Software Approaches in Distributed Modeling and Simulation, February 2001.

Luke S., L. Cioffi-Revilla, K Panait, K. Sullivan and G. Balan 2005. MASON: A multiagent simulation environment. SIMULATION, 81:517-527.

Minar N., R. Burkhart, C. Langton and M. Askenazi. 1996. A toolkit for building multi-agent simulations. Santa Fe Institute.

Mockapetris, P. V. 1987. Domain names - concepts and facilities. Obsoletes RFC0973, RFC0882, RFC0883. See also STD0013 Updated by RFC1101, RFC1183, RFC1348, RFC1876, RFC1982, RFC2065, RFC2181, RFC2308. Status: STANDARD.

Morse, K. L. 1996. interest management in large-scale distributed simulations, University of California, Irvine. 96-27.

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

Nicholas, R. J. 2000. "On agent-based software engineering." Artif. Intell. 117(2): 277-296.

Patrick, F. R. and F. R. George 2003. Next generation modeling III - agents: Spades --- a distributed agent simulation environment with software-in-the-loop execution. 35th conference on winter simulation: driving innovation, New Orleans, Louisiana, Winter Simulation Conference.

Riley, P. F., G. F. Riley. 2003, December. SPADES – a distribution management in distributed simulations. In Proceedings of 2003 Winter Simulation Conference (WSC-2003), 817-825.

Tacic, I., and R. Fujimoto. 1998. Synchronized data distribution management in distributed simulations. In Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS-1998). 108-115.

Vulov, G., T. He and M. Hybinette. 2008. Quantative assessment of an agent-based simulation on a Time Warp executive. Winter Simulation Conference, Miami, FL.

Uhrmacher, A. M., P. Tyschler and D. Tyschler. 2000. Modeling and simulation of mobile agents. Future Generation Computer Systems 17 (2): 107–118.

Wang, L. S. J. Turner and F. Wang. 2003. Interest management in agent-based distributed simulations. In Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2003), 20-29.

Xiong, Y., M. Hybinette and E. Kraemer, Transparent and adaptive computation-block caching for agent- based simulation on a PDES core. 2008 Winter Simulation Conference (WSC-2008)