Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. Hill, L. Moench, and O. Rose, eds.

QUANTATIVE ASSESSMENT OF AN AGENT-BASED SIMULATION ON A TIME WARP EXECUTIVE

George Vulov, Tianhao He and Maria Hybinette

Computer Science Department
University of Georgia
Athens, GA 30602-7404, USA

ABSTRACT

We recently introduced SASSY, the design for a hybrid simulator that provides an agent-based API atop a PDES kernel (Hybinette et. al. 2006). Our hypothesis is that a design like SASSY offers the advantages of an agent-based paradigm for the application developer, but also provides the performance advantages of a PDES kernel. Since the time of our initial publication, most aspects of SASSY's design have been implemented, and we are now assessing our hypotheses e.g., (He and Hybinette 2008). In this paper we investigate performance advantages for a simple agent-based application on SASSY. In most cases, agent-based simulation environments are configured using a time-step approach, where the simulation proceeds in discrete steps. In this paper we evaluate the performance of a simple application running in a traditional time-step simulation, and also its performance when running on SASSY with PDES support.

1 INTRODUCTION

Multi-agent simulation is becoming more prevalent in different areas of research, such as 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. The **S**calable **A**gent-based **S**imulation **Sy**stem (SASSY) project aims to leverage advances in the field of Parallel Discrete Event Simulation (PDES) to provide an agent-based API but with the scalable 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.

SASSY's PDES kernel, based on the Time Warp synchronization algorithm (Hybinette et. al. 2006) has been completed. Figure 1 provides an illustration of SASSY's design and implementation. 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.

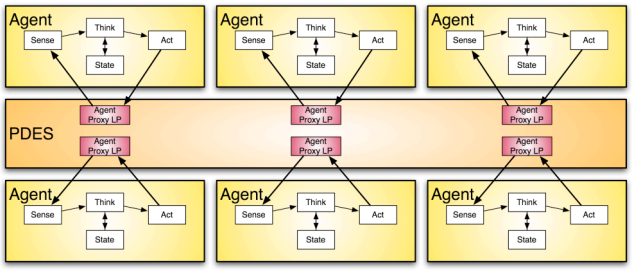

Figure 1: SASSY Architecture.

Our recent work reported in this paper has focused on designing and implementing the agent-based middleware, along with a simple multi-agent simulation application to test its correctness and performance. 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. The SASSY middleware implements the agent environment as a 3D Euclidean space. While the agent environment is not fully generalized, assuming the environment is Euclidean is not very restrictive and allows for better techniques for parallelization. The approach to distributed agent-based simulation presented here can easily be implemented for n-dimensional Euclidean space, although 3-space was chosen for the current implementation.

The agent-based paradigm dictates a number of requirements for the PDES kernel, which has driven us to add a few more advanced PDES techniques into the SASSY kernel. We discuss these additions in the sections below. First we provide some background on PDES simulation systems and other related work.

2 BACKGROUND AND RELATED WORK

Our architecture, previously described in (Hybinette et. al. 2006) is designed to leverage the efficiency, speed and parallelism available in discrete event simulation (DES) systems to support agent-based modeling (ABM). We use a *standard* parallel discrete event simulation (PDES) kernel paired with middleware to provide an agent-based paradigm for the simulation application developers (e.g., like TeamBots (Balch 1998), Swarm (Minar et. al. 1996), Mason (Luke et. al. 2005) or Player/Stage (Gerkey et. al. 2003).

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, 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 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 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, and lazy cancellation and re-evaluation.

Our approach leverages lazy cancellation, a technique used in optimistic simulators to improve the performance of rollbacks. Lazy cancellation delays secondary rollbacks and caches the results of the original rollback in anticipation of reusing the results and thus avoiding recomputation (Gafni 1988). We now present our algorithm.

3 APPROACH

3.1 Exploiting Spatial Locality in a Distributed Environment

As described in (Hybinette et. al. 2006), 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 dis-

tribution 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.

3.2 Optimizing Relayed Communication of the Shared Environment

An implementation of the above design yields IMLPs that do little but relay messages between proxy LPs; it may be faster to let proxy LPs communicate environmental updates directly to each other using a peer-to-peer mechanism (He and Hybinette 2008). 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 2. The 'region' is light and agent F is in the region at time steps 2, 3, 4 and 5 at a wall clock time before S. S enters the region at simulated time 2.

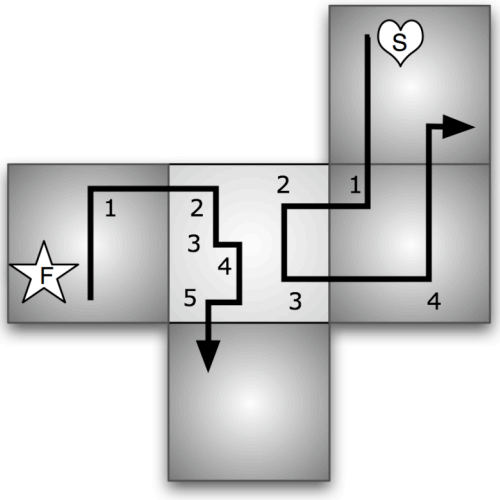

Figure 2: Example of an agent F entering a light color region and an agent S entering the same region.

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 cancelling messages it sent to other subscribers). This is illustrated in the lower image in Figure 3, here S's message at simulated time stamp 2 rolls back F's messages with later time stamps.

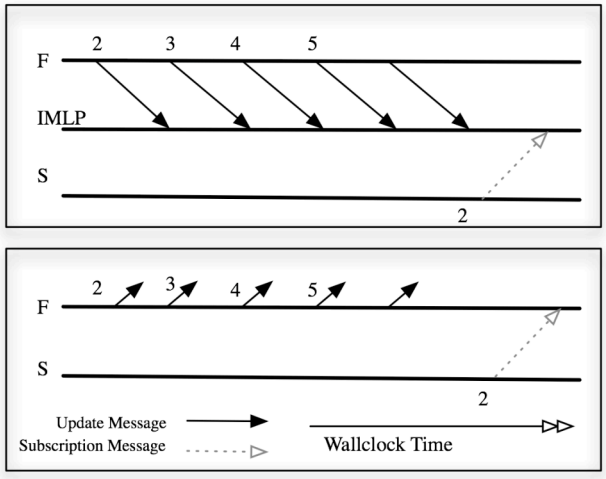

Figure 3: Message Progression as a slow-processing agent enters a region where another agent has passed

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 cancelling message updates to other subscribers). This scenario is il-

lustrated in the top image of Figure 3. Here the IMLP forwards the buffered messages with time stamps later than S's subscription message. Note this avoids rolling back and cancelling 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).

3.3 Implementing Selective Lazy Cancellation in Java

When an LP rolls back to a virtual time, the messages it sent before that virtual time have to be cancelled. Lazy cancellation waits until the LP processes back to its original time and only cancels the anti-messages that were not re-generated. This is normally implemented by storing all the rolled back outgoing messages. As the LP sends new messages, every new message is compared bitwise to the rolled back messages. If there is a match, then both new and old messages are discarded. However, the SASSY PDES kernel is implemented in Java, where direct memory access for bitwise comparisons is not allowed.

In order to implement lazy cancellation in Java, some method must be devised so that the kernel can compare the contents of two messages and decide if they are the same. One option is to force all PDES messages to implement Java's *Comparable* interface, which would ensure that all messages have an explicit comparison method. Unfortunately, this approach would obligate the PDES application developer to write a new method for every type of message being used. Moreover, a programming error by the application programmer can compromise the correctness of the PDES kernel. The SASSY PDES kernel resolves this issue by leveraging the serialization feature of Java. Once a message is serialized, it is available for bitwise comparison. The kernel already uses serialization to transfer messages over the network; so using serialized messages for lazy cancellation did not add additional overhead.

The SASSY kernel contains an additional option in its PDES API: turning on lazy cancellation for some LPs and not others. Thus, the agent middleware can enable lazy cancellation only for interest manager LPs, where its benefits are known. In the future we plan to modify the kernel to allow PDES applications to apply lazy cancellation on a per-message basis. There exist many situations in which the PDES application can judge the benefits of lazy cancellation and provide a hint to the kernel. For example, a message that simply requests information from an LP (i.e. a query) can be marked for lazy cancellation. Note that the correctness of the simulation would not depend on the lazy cancellation hints; they are simply a performance improvement.

3.4 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. 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 $\Delta = 10$ 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 time 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.

3.5 Ensuring Correctness and Repeatability of Simultaneous Messages

Implementing the agent-based paradigm on a PDES simulator invariably induces PDES messages with simultaneous timestamps. If the SASSY middleware is hosting 100 agents, each thinking every time interval, there will be 100 simultaneous SENSE events for every virtual time. In a serial time-stepped simulator, the agents would always take turns moving in a fixed predictable order. Therefore, the SASSY middleware must have a way to specify the order in which SENSE events are processed.

Consider two agents, A and B, moving about in the same IMLP region. Both proxy LPs have scheduled their next SENSE event with timestamp 20. In order to provide consistent result SASSY's middleware selects a deterministic ordering of the agents. If the ordering calls for an agent A to process before agent B, agent A's SENSE event runs first. Agent A may then modify the environment, sending an UPDATE message to the IMLP. The IMLP then reflects the UPDATE message to Agent B. To ensure correct behavior, Agent B has to be notified of A's modifications to the environment before its SENSE event, which is scheduled with timestamp 20. This example illustrates that messages sent by proxy LPs to the IMLP and messages relayed from the IMLP must both be 0-lookahead events.

The SASSY agent-based middleware thus produces both simultaneous events and a number of 0-lookahead events for each discrete time interval simulated. The discrete event simulation community has long recognized that improper handling of simultaneous events can lead to incorrect behavior (Fujimoto 2000) or inconsistent results (Wieland 1997). SASSY's PDES kernel implements a tiered tie-breaking approach described in (Fujimoto 2000). To ensure 0-lookahead messages are executed in their dependency order, every 0-lookahead message includes some identifying information about its ancestor messages. If two PDES messages have the same time stamp and are independent, then the correct ordering depends on the simulation model. Therefore the SASSY kernel allows for an application-supplied *Comparator* object that orders independent simultaneous events. Finally, if the PDES application does not break the tie, the kernel uses extra LP fields (Fujimoto 2000) to deterministically schedule an event first, for the purpose of repeatability.

The kernel measures described ensure that simultaneous events are processed in such a way that the PDES simulation progresses and is repeatable. In addition, the SASSY middleware installs its own tie-breaker to maintain the correctness of the agent-based simulation. Messages received by an IMLP are processed by the agent's movement order. For instance, if two agents send an UPDATE message to the IMLP with the same time stamp, the agent which processes first will have their UPDATE message applied first. Messages from the same proxy LP to an IMLP are processed in the order { LEAVE, UNSUBSCRIBE, SUBSCRIBE, ENTER, UPDATE }. This ordering ensures that an agent is properly subscribed and registered with a region before sending any updates to the region. When an agent's proxy LP encounters simultaneous messages, it processes the reflected messages of earlier agents before its SENSE event and messages of later-moving agents after its SENSE event. Messages reflected from the same agent are processed in the order { LEAVE, ENTER, UPDATE } to ensure consistency of the shared environment.

It must be noted that even though the SASSY middleware maintains a notion of the order in which agents process, it is simply to present a deterministic world view which is consistent with serial time-stepped simulation. Distributed agents freely process out of the specified order when their actuating regions do not overlap. The SASSY tie-breaking system is also easily extensible to provide a certain level of fairness to the simulation. If there are n agents in the simulation, there are n! possible orders in which they can process. The tie-breaker can be modified to enforce a different processing order for each simulation time. For example, three agents could process in the order 1 2 3 the first time step, then in the order 2 3 1 in the second time step, then in the order 3 1 2, etc. Therefore in addition to offer performance advantages over a serial time-stepped simulator, SASSY has the potential to offer more modeling flexibility.

4 PERFORMANCE

4.1 The Agent-based Application

The multi-agent simulation used for performance testing consists of a number of bouncing balls. Each ball is an in-

dependent 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.

4.2 Distributed Performance of SASSY's Middleware

In our first performance tests, we chose to test the scalability of SASSY, with regard to the number of agents and the number of machines SASSY runs on. The simulation environment had 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.

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

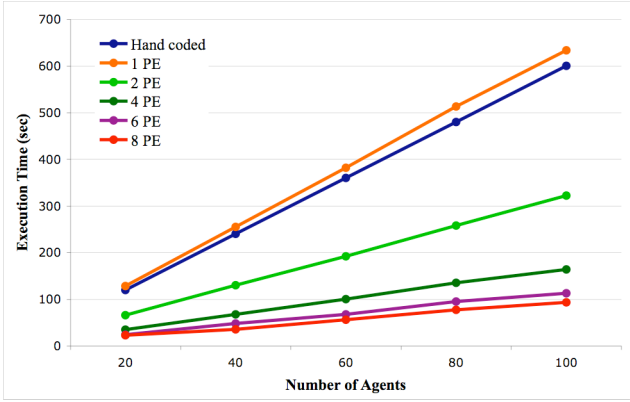

Figure 4: Execution time with Growing Number of Agents

Figure 4 illustrates he performance of SASSY as the number of agents increases and also as the number of PEs changes. When executing on only one Processing Element (PE), the overhead of the SASSY middleware and the PDES kernel makes the SASSY simulation slightly slower than the hand-coded time-stepped simulator (Figure 4). Fortunately, the overhead is very slight; it is 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.

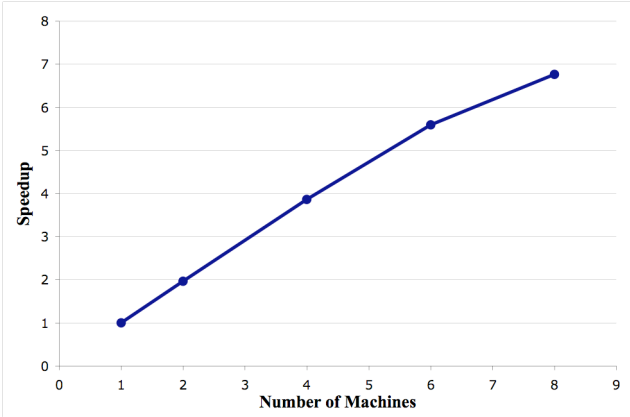

Figure 5: Distributed Execution Speedup
(100 Agents, 300 msec deliberation)

Figure 5 shows how 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 accomplishing 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.

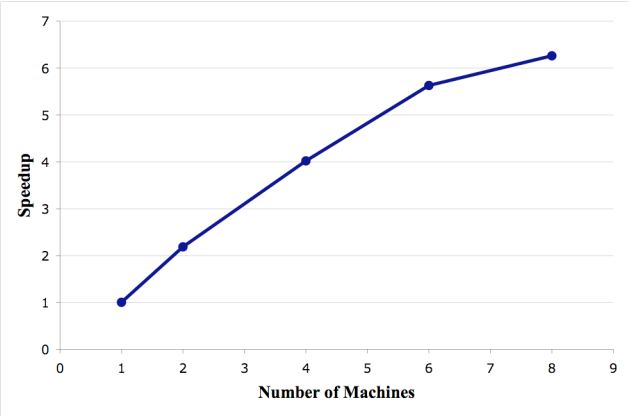

Figure 6: Distributed Execution Speedup (3000 Agents, 5 msec deliberation)

The speedup growth in Figure 6 is clearly less linear than the speedup growth in Figure 5; 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.

4.3 Performance with Agents with Varying Sensing Distances

In our discussion of relayed vs. 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 quite common; for example a robot can have a video camera with a wide view but have 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 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.

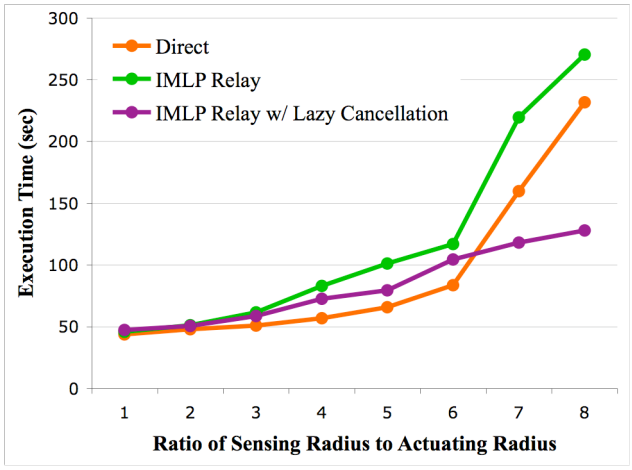

Figure 7: Execution Performance with Increasing Agent Sensing Distance

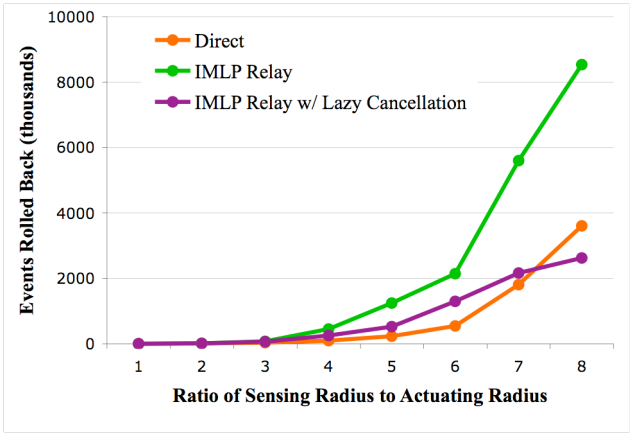

Figure 8: Number of Events Rolled Back with Increasing Agent Sensing Distance

Figures 7 and 8 describe results from the same executions of the simulation. 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 relay 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.

5 SUMMARY 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 as 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 throughput, 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 9). The agent-based modeling API provided by the middleware can also be improved, accommodating additional use cases.

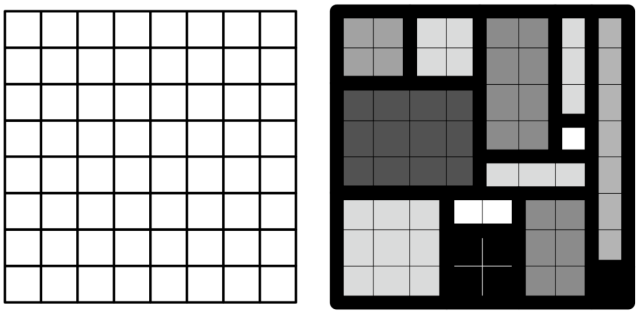

Figure 9: Current IMLP Regions vs. More Flexible IMLP Regions

For the testing application, in future work we plan to develop more comprehensive and realistic agent-based tests cases. One agent-based simulation will contain actually 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

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

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. October. Parallel discrete event Simulation. *Communication of the ACM* 33 (10):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. In *Proceedings of the SCS Multiconference on Distributed Simulation*. 19:61-67.

Gerkey, B., R.T. Vaughan, and A. Howard, 2003. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the Interna-*

tional Conference on Advanced Robotics, 317–323. Coimbra, Portugal.

Haumacher B and M Philippsen 1999. More Efficient Object Serialization. In Proceedings of the International Workshop on Java for Parallel and Distributed Computing of Lecture Notes in Computer Science Volume 1586, pages 718-732.

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.*

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).

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.

Pollack, M. E., and M. Ringuette. 1990. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI Press, pp. 183-189.

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.

Steinman, J. S. and J. W. Wong. 2003. The SPEEDES persistence framework and the standard simulation architecture. *Parallel and Distributed Simulation, 2003 Proceedings*. Volume 10-13 (June 2003): 11 – 20.

Taboada, G.L. C. Teijeiro, J. Tourino, 2007. High Performance Java Remote Method Invocation for Parallel Computing on Clusters. In 12th IEEE Symposium on Computers and Communications (ISCC-2007): 233-239.

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.

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.

Wieland, F., 1997. The threshold of event simultaneity. In Proceedings of the 11th Workshop on Parallel and Distributed Simulation (PADS-1997). 108-115. 56-59.