ADAPTIVE MESSAGE CLUSTERING FOR DISTRIBUTED AGENT BASED SYSTEMS

BY

ABHISHEK GUPTA

(Under the Direction of Maria Hybinette)

ABSTRACT

Many agent-based simulation kernels rely on message passing in their core implementation. As the number of agents in a simulation increases or as the complexity of their communication expands the number of messages can increase exponentially. This is troublesome because the message content itself may be quite small, while the overhead, including message headers can dominate bandwidth and processing time. In these cases message passing becomes bottleneck to scalability: The overhead of message exchange may saturate the network and degrade performance of the simulation. One approach to this challenge that has been investigated in related networking and simulation research centers on combining or "piggybacking" multiple small messages together with a consolidated header. In many applications performance improves as larger, but fewer messages are sent. However, the pattern of message passing is different in the case of agent-based simulation (ABS), and this approach has not yet been explored for ABS systems.

In this work we provide an overview of the design and implementation of a message piggybacking approach for ABS systems using the SASSY platform. SASSY is a hybrid large scale distributed agent-based simulation system that provides an agent–based API to a PDES kernel. We provide a comparative performance evaluation for implementations in SASSY with a combined RMI, and shared memory message passing approach, and RMI only. We also show performance of our new adaptive message clustering mechanism that clusters messages when advantageous and avoids clustering when the overhead of clustering dominates.

INDEX WORDS: Agent Based Simulation, Distributed Simulation, Piggybacking, SASSY

ADAPTIVE MESSAGE CLUSTERING FOR DISTRIBUTED AGENT BASED SYSTEMS

BY

ABHISHEK GUPTA

B.E. BHARATI VIDYAPEETH UNIVERSITY, INDIA 2006

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

2010

ADAPTIVE MESSAGE CLUSTERING FOR DISTRIBUTED AGENT BASED SYSTEMS

BY

ABHISHEK GUPTA

Major Professor: Maria Hybinette

Committee: Kang Li

Lakshmish Ramaswamy

Electronic Version Approved:

Maureen Grasso

Dean of the Graduate School

The University of Georgia

December 2010

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Maria Hybinette for all her help and encouragement. Without her guidance and support this work would have not been completed. Her suggestions and feedback have made the completion of thesis possible.

I would also like to thank Prof. Lakshmish Ramaswamy and Prof. Kang Li for their time and cooperation. I would like to thank and congratulate the other members of the SASSY research group who designed a stable and powerful framework. I would especially thank Tianhao He for helping me understand SASSY during the initial days of my research. Tianhao He implemented the Interest Manager mechanisms for the Multi Agent Based Simulation layer in SASSY.

TABLE OF CONTENTS

CHAPTERS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Agent Based Simulation **(ABS)** systems has gained significance in recent years because they provide a more natural way for simulation application designers to encode problems than other simulation methods such as discrete event simulators. ABS systems are applied in the study of multi-robot systems, social animal behavior, and automobile traffic by researchers in various areas ranges from robotics to biology for examples: MASON (Luke et al. 2005), Player / Stage ((Riley and Riley 2003; Gerkey et al. 2003), SWARM (Langton et al. 1995).

The simulation kernel implements interactions between agents such has communication, sensing and acting typically by message passing but details of this mechanics are hidden from the application programmer by the agent based application programmer interface. Because message passing is central to an ABS system, it may also become a bottleneck. The efficiency of message passing in an ABS is the focus of this research.

1.1 Problem Statement

The implementation of an ABS is challenging for complex systems in which the agents are mobile and in a constant motion. These agents communicate among themselves by exchanging messages. Furthermore, the simulation kernel often implements agent's sensors also by message passing (even though this may not be revealed to the agents themselves). Accordingly a large scale Agent Based Simulation often involves the exchange of a large number of small messages that may saturate the network and degrade performance. An example of what a network of

communicating agents may look like is illustrated in Figure 1 (interactions are denoted by white rays).

Figure 1: Communication patterns for agents in a simulated environment**.**

(Picture credit: http://www-f1.ijs.si/~tadic/COST_MP0801/?page=members)

The topology of communication between agents depends on the simulation scenario. For instance, agents might have a reduced sensing range (e.g., ants sensing range is limited to approximately the length of five ants, see Figure 2) in comparison to the size of the entire simulation environment. In this case the communication topology is simplified significantly from the fully connected case, which is often assumed in simulation implementations.

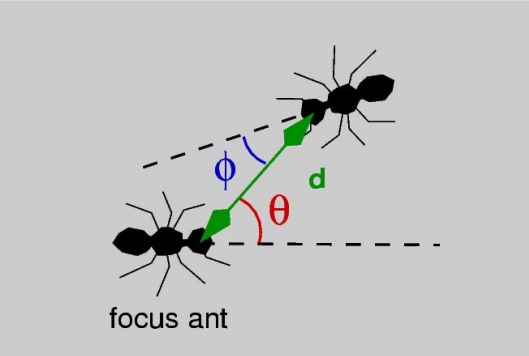

Figure 2: Interactions between ants *(*Figure credit: Balch et al, 2005*)* **

Our approach is to exploit limitations in communication observed in the 'real system' simulated (e.g., ants limited sensing range or obstructions in the terrain such as trees and mountains), including being able to respond to dynamic changes in the communication patterns in the simulation.

As the complexity of ABS scenario increases (perhaps by a larger number of agents) the number of messages exchanged may increase exponentially. Performance may begin to degrade and result in a significantly increase in execution time. Each message sent and received incurs overhead (e.g., processing headers, extracting messages from the network and payload), and this work investigates methods to reduce the cost of exchanging messages and reduce redundant information. The aim of this research is to test the hypothesis that clustering (or aggregating) messages before sending them can reduce the overall communication cost and improve performance. We believe a clustering mechanism can improve performance by aggregating multiple messages into a single message.

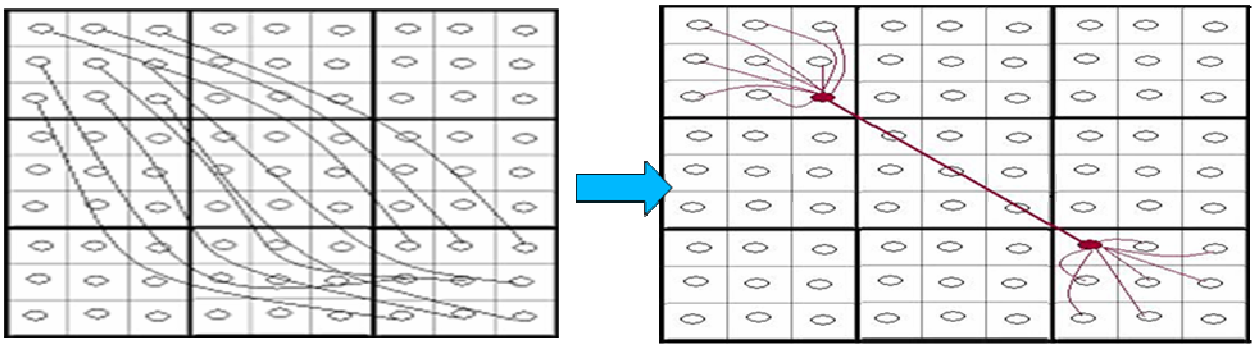

Figure 3: Message clustering approach

More specifically, in this research we investigate factors that impact the effectiveness and performance of message clustering or aggregation of messages to speed up the performance of distributed agents based simulation and provide a quantitative analysis. We also propose a new adaptive scheme that dynamically determines whether and when to utilize message aggregation and determine the number of messages clustered, and we show its performance and compare it with a non-adaptive clustering approach and an unclustered approach.

Our empirical study and new adaptive mechanism is implemented in the **S**calable **A**gent-based **S**imulation **Sy**stem (SASSY), a java based distributed simulation kernel developed by our research group (Hybinette et al. 2006).

SASSY is aimed to leverage advances in the field of Parallel Discrete Event Simulation (PDES) for Agent Based Simulations. SASSY was designed and developed in order to overcome the scalability issues that exist in current Agent Based frameworks. The approach discussed in this research and in SASSY inserts middleware between the Application Programming Interface (API) and a standard PDES simulation kernel (SASSY)

Our implementation focuses on the Interest Management Logical Process (IMLP) mechanism implemented in SASSY (He et. al 2008). Each agent subscribes to a particular Interest Manager Logical Process or IMLP using a publish/subscribe protocol. The agents transmit messages with variably sized payloads to agents in possibly different regions via IMLPs, which may reside on remote machines.

In the clustering approach described here, messages are *clustered* at local IMLP before being forwarded to its remote IMLP and then transmitted to the receiving agent. Once the

clustered message has reached its IMLP it is then un-clustered and forwarded to its respective destinations (See Figure 3 above)

In this work we use discrete gridded world in order to exploit our communication algorithm and leverage constraints in the application model such as ants cannot see every other ant in the environment). The sample space has been divided into regions (cells) with each of the mobile agents being subscribed to an IMLP in the corresponding cell. This way we try to reduce to some extent the unwanted exchange of messages that could have happened in general peer-to-peer network. We measure performance in terms of speedup gained by implementing this communication topology in SASSY.

In the next section we review background, including different simulation techniques and agent based simulation, we then review related work both from artificial intelligence, and simulation community, and then we present the implementation in detail and we conclude with results and future work.

CHAPTER 2

BACKGROUND

Simulation is a computer program that attempts to simulate a particular system or behavior of an entity. Simulations have found its application in varied branches of studies like mathematical modeling, computational physics and modeling, biology etc. One area where simulation has contributed to a great extent is medical sciences. Medical simulation is a branch of simulation that educates people in the area of medical fields like human patients, study and analysis of various human diseases and their impact on human behavior. Another major area where simulation has found its use is in Military and emergency responses. United States government spends billions of dollars to promote simulation for space exploration, computer advancements, and military trainings, development and training of flight simulators.

The work presented in this thesis is implemented within the **S**calable **A**gent-**B**ased **S**imulation **S**ystem (SASSY) (Hybinette et al. 2006) developed at the Distributed Simulation lab at the University of Georgia. SASSY is implemented in java and uses a standard parallel discrete event infrastructure.

In discrete event simulation, the simulation state changes only at discrete intervals of time. When changes in state happen at constant intervals in time it's called as time stepped simulation, when changes in state happens sporadically by the trigger of an events it's called event driven. In the latter the simulation receives an event, and it advances it's time to the time-stamp of the current event and marks the message as processed. Processing a message may or may not lead to generation of new messages. When discrete event simulators are distributed across processors, possibly on remote machines it is termed Parallel Discrete Event **S**imulation

(PDES). The major advantage of PDES is reduction in overall computation time of the simulation system.

Parallel simulation is composed of distinct components called logical processes or LPs. Each LP models some portion of the physical system. For example in an air traffic simulation an LP might represent an airport. The logical processes may be mapped to different processors. A change in state is accomplished by an event. The messages sent from one LP that may request the destination LP to change its state leads to change in state of an event.

For example in air traffic simulation processing a departure event may result in scheduling a new arrival event at another airport. 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.

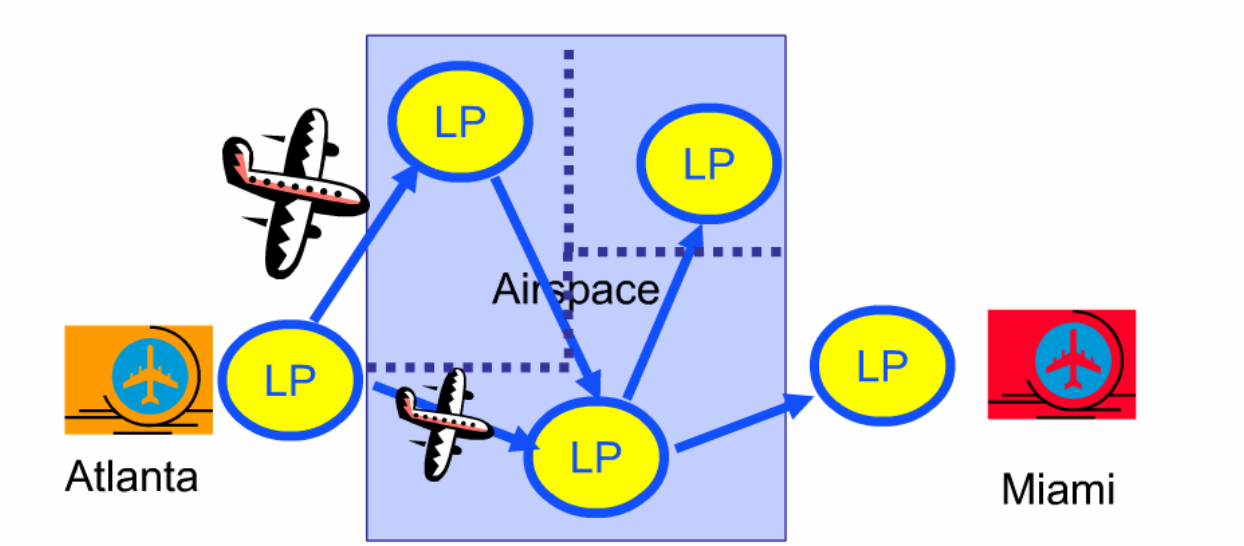

Figure 4: Parallel Discrete Event Simulation (Figure credit: Fujimoto (2000))

For example in Figure 4, depicting an air traffic simulation with airports Atlanta and Miami airports, these are mapped to LPs that are a part of the simulation system. The above figure shows some action (time stamp: taking off; landing) happened in the physical system.

Even though the figure above shows one to one mapping between a physical process in the simulation (an airport) and the logical process (in the simulation kernel) however in most PDES kernels processes in the real system can be mapped to multiple logical processes.

Events within each logical process must be processed in time stamp order. This ensures that the parallel simulation will produce exactly the same results as a corresponding sequential simulation. There are two major categories of synchronization protocols in PDES: conservative (Chandy and Misra 1981) and optimistic (Jefferson and Sowizral 1985). Conservative protocols process events when it's safe to process and waits or blocks when it is not safe. Optimistic protocol in contrast process events immediately and recovers from causality errors by rolling back events and reprocessing the events including the 'straggler' that caused the initial causality error in the correct order accounting for the straggler. SASSY uses an optimistic synchronization protocol although our communication mechanism should theoretically work for a conservative kernel as well.

2.1 The Agent Based Paradigm

PDES systems are not always well suited for agent based applications like biological systems ants, bees, robot systems etc. Agent based community expects agents as objects that can move around in an environment similar to exchange of messages in Discrete Event Simulations along with the computational abilities. In most of PDES simulations LPs are static. Network routers,

and airports can be visualized as good examples of such agents. In contrast in most agent based simulations the agents move around, e.g., ants and bees or robots. In general, ABM researchers expect their agents to (Riley and Riley 2003, Balch 1998, Gerkey et al. 2003, Hybinette 2001):

- **Use the Sense-Think-Act cycle**: agents sense their environment, analyze the action that needs to be taken and then act accordingly. This represents the computational paradigm for agents.
- **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 the reasons stated above a number of Multi Agent Simulation systems and multi robot systems researchers have devised their own simulation systems for their research, however these simulators lack in performance. These limitations prevent MAS researchers from investigating systems with thousands 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.

2.2 SASSY Framework

The message clustering/ piggy backing technique was implemented over SASSY framework. In this framework, a faster than real time simulation runs on a number of computing nodes, possibly

distributed over a network. Faster than real time simulation allows models to advance ahead of the corresponding wall-clock time.

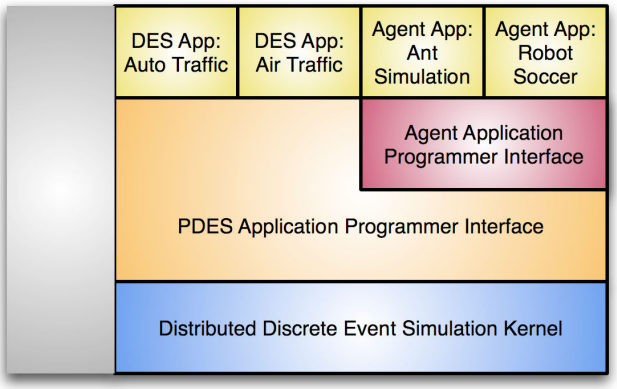

Figure 5: SASSY: The (S)calable (A)gent(s) (S)imulation (Sy)stem

(Figure credit: Hybinette et. al, 2006)

In a standard physical agent model, an agent senses its environment, considers what to do, then acts. This is frequently referred to as the sense-think-act cycle (Riley and Riley 2003, Uhrmacher et al. 2000, Logan and Theodoropoulos 2001), see Figure 6.

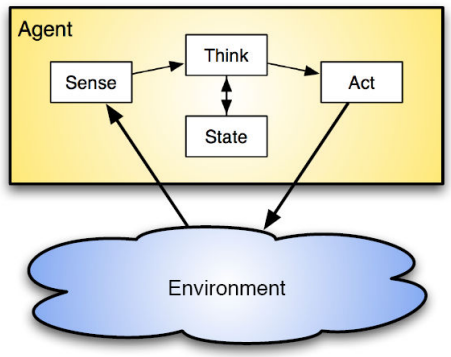

Figure 6: The physical agent model (Figure credit: Hybinette et. al, 2006)

An 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).

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).

SASSY provides a PDES API (application programmer interface), as well as an Agent-Based API that is implemented as middleware on top of a PDES kernel. SASSY is implemented in Java, and therefore benefits from object oriented system design. The kernel provides an abstract class Logical Process that implements the features of a generic logical process. These methods are implemented as private; the application programmer does not need interact with them. Methods requiring application specific implementation are designated abstract and are to be implemented by the application programmer. The LPs schedule events both remote and local (including events to itself) via messages. A SASSY message is a Java object that specifies the destination of the message and is sent via the sendSimMessage () method. SASSYs agent-based modeling API is implemented as middleware between the PDES kernel and the agent-based model application code. Each agent is provided a PDES proxy LP that serves to process and create messages. Figure 7 shows how this works for one agent.

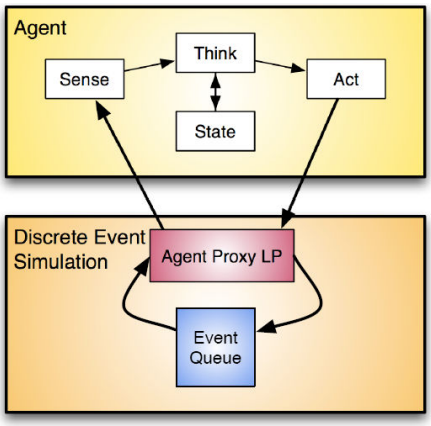

Figure 7: An LP in the PDES System Serves As a Proxy for a Simulated Physical Agent.

(Figure credit: Hybinette et al, 2006)

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, and then the IMLP reflects those messages to other interested agents.

CHAPTER 3

RELATED WORK

There has been some work done in the area to reduce communication costs in both the networking and simulation community. Our work presented here is inspired by previous work ant it differs mainly in that it leverages knowledge available from the application layer and that is adaptive both in terms of number messages clustered and that it avoid clustering on the fly when it is determined that it is not advantageous. It is also unique in that it runs in a distributed agent based simulation paradigm driven by a discrete events simulation executive transparent from the application programmer.

In the networking research community, Roy Friedman and Robert van Renesse of Cornell (Renesse et al.1995) compared the throughput and latency of two protocols (Tomfc and Dysfc) with and without message clustering. Their technique involved buffering the application messages for short period of time before being sent to their respective destinations. Their study showed that message clustering had some impact over the performance of the system. They used the application message buffering technique for this purpose. This include buffering messages for a short period of time then clustering all the messages that have been buffered and sending them as a single packed message. They found that packing of messages improved both latency and throughput to two orders of magnitudes. The idea was that if each message is delayed but only for a short period of time the throughput is increased dramatically. Also this packed message helps in reducing the contention on the network and also reduces the load in terms of the processing capacity and speed on the receiving processors considerably. So this reduced the user-to-user latencies considerably in a highly communication intensive environment.

The reasons for this improved performance were attributed to the fact that packing of messages reduces the total number of bytes sent as for each clustered packet there exists only one header instead of a header of a message. Piggybacking of messages leads to a smaller number of messages that needs to gain access to Ethernet so there is less contention for network hardware as well. Further on the hardware level, a receiving CPU only handles one interrupt for the piggybacked messages instead of an interrupt for each message packaged in the single message.

Message aggregation techniques have also been proposed for sensor networks, e.g., AIDA, the application independent data aggregation approach proposed by (He et al. 2004). Here application layer is ignored and decisions are made on lower network layers. It is adaptive approach and it has shown promising results in simulation. Other adaptive message aggregation protocols for sensor networks include RAP (Lu et al. 2002) and SPEED (He et al. 2003). They use the neighborhood (nodes) information like network congestion and traffic to make an informed decision about the message routing and aggregation. SPIN (Heinzelman, 1999) makes adaptive decisions to participate in data aggregation based on the cost of communication.

Other related techniques focuses on aggregating data (instead of messages) includes, the Center at the Nearest Source (CNS), where data is aggregated at the source nearest to the destination (Intanagonwiwat et al. 2002), here data is aggregated at the source nearest to the destination, in Shortest Path Trees (SPT), data is propagated along the shortest path from source to destination and aggregated at common intermediate hops along the way.

Clustering of messages for ABS in PDES is a relatively new concept. Takahashi and Mizuta (Takahashi et al. 2006) addressed the message transmission costs in a large and complex agent based simulation system on a large multi-node super computer, BlueGene, by clustering

(and re-clustering) heavily communicating agents on the same node so that these agents can communicate via shared memory instead of remote message passing. They essentially used load-balancing approach to reduce communication cost. They show promising performance result for 2 and 4 remote nodes connected via gigabit Ethernet. Our approach differs in that we use message clustering instead of load balancing to reduce communication cost. We do leverage shared memory to communicate when agents reside on the same node and in future work we plan to combine message clustering with load balancing to further improve performance. A similar protocol in the networking community that promotes local communication is LEACH (Heinzelman et al.2000), where LEACH is a high layer protocol that provides clustering and local processing to aggregate sensor data and reduces global communication.

Agents in our application use a publish and subscribe protocol between agent and Interest Managers (IMLPs). Each IMLP manages a subset of agents depending on where the agents reside in the environment (He et al. 2008). Logan and Theodoropoulos also proposed a related approach but here world state is maintained by a single environment LP that is currently centralized however in SASSY there is no centralized state and each agent keep track of its own state. Unlike Takahashi and Mizuta agents load balancing approach to reduce communication we piggyback messages sent by agents to other heavily interactive agents. Our mechanism is implemented in a fully distributed environment and has been evaluated up to 10 remote processors and 1200 agents. We control the degree of coupling of agents through IMLPs. This also impacts degree of coupling and decoupling that IMLPs can manage (discussed in detail in chapter 5). The message sizes varied from 100 bytes to 1KB for up to 1000 agents for 64 IMLPs. In future work we hope to combine message clustering, data aggregation and load balancing as proposed in the networking and sensor networks.

CHAPTER 4

CLUSTERING APPROACH

In order to evaluate the affect of various message-clustering parameters on performance, we implement our own message-clustering scheme in SASSY (a java based agent based simulator developed by our research group at the University of Georgia) and evaluate the scheme experimentally.

In SASSY, agents are represented by a proxy logical process (LP). These proxy LPs maintain the most updated version of the environment visible to the agent. The SASSY middleware decomposes the sample space or the environment into a grid of cells for efficiency and scalability reasons. An Interest Manager Logical Process (IMLP) manages a cell or set of cell in the gridded environments. The IMLP implements a subscribe/publish system for the proxy LPs to ensure that all agents have a consistent global view of the environment unlike other Agent Based Systems that use a centralized global view of the environment for all the agents (Riley and Riley 2003, Gerkey et al. 2003). Adjusting through a parameter at run time can control both the size of the cells (area covered in the environment) or the number of cells managed by IMLP. Each agent is part of a cell in the gridded world that is managed by an IMLP. So effectively every agent will be having at least one IMLP mapped to it. We leverage this concept to implement the message-clustering algorithm in SASSY, but first we will describe the message types available in SASSY briefly below.

A Proxy LPs can send following types of messages to an IMLP:

- SUBSCRIBE
- UNSUBSCRIBE
- ENTER
- LEAVE
- 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. On similar lines **UNSUBSCRIBE** messages removes an observing agent from the IMLP list. 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. 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. An Proxy LPs to notify other agents' proxy LPs of changes in the observable environment uses an **UPDATE** message. An **UPDATE** message is relayed by the IMLP to all subscribers.

We added three new types of messages to the Proxy LP to implement our message clustering mechanism:

- CLUSTER
- UNCLUSTER, and
- PING

CLUSTER is the message sent by IMLP of one cell in the grid to and IMLP of another cell that holds the clustered messages of frequent interacting agents. The messages in this clustered packet can be of varying payload size. In this research we have varied the payload size ranging from 1 byte to 1,000 bytes. The **UNCLUSTER** message is sent by an IMLP to its subscribed agents when it receives a clustered message packet. This leads to unclustering of the received packet and the IMLP then forwards the message to its respective destination by local communication (typically shared memory). The clustered message is created using a hash map data structure at every IMLP that holds the ID of the agent and message(s) as a key value pair. This helps routing the messages on unclustering to their appropriate destination. Messages destined to an agent belonging to the same or different cell issue a **PING** message. **PING** is control message to the IMLP that is implemented as a wrapper informing the IMLP that messages beginning here and onwards are candidates to be clustered.

In practice the mechanics of agent communication in SASSY is that IMLPs relays messages to the respective IMLP. The IMLP in turn forwards the message to the corresponding agent by forwarding it to the respective IMLP. In previous work we evaluated both direct communication and relay communication, however relaying showed better performance (Vulov, He et al. 2008)

In this work we assume communicating is dynamic in that agents in general do not broadcast messages to every other agent but multicast messages to a subset of agents in the environment (this is also true in biological systems such as ants that only communicate with ants or agents within its sensor range). This means that at a given instant of time all agents would not be communicating with all the other agents present within the grids. The agents located in a cell

at a particular instant will send messages to immediate neighboring IMLPs (cells) only. This ensures the message transmission is multicast to other communicating nodes. Agents will send messages to its respective IMLPs and IMLPs will in turn forward it to immediate neighboring IMLPs this avoids broadcast. Hence, we discard the idea of broadcasting of messages. Thus we have assumed that agents belonging to a grid always communicate frequently with agents belonging to its neighboring regions (e.g., subscribers)

To evaluate whether clustering is advantageous and its overhead we implemented two algorithms: (1) Fixed Clustering and (2) Adaptive Clustering. We compared these algorithms against a non-clustering approach.

4.1 Message Transmission in SASSY and in the Clustering approaches

In order to evaluate message clustering we varied the message pay load. In SASSY an agent is initialized by the **`appInitializeLP()`** method. We added the variable payload message sent by every agent for other agents here. Java random function was used to make sure the payload varies for every agent with every iteration or move of the simulation. The seed value in the below function was decided upon depending on the range of payload message we wish to transmit.

```
randomGenerator.nextInt( seedValue )
```

Once, the agent subscribes to an IMLP by sending SUBSCRIBE, ENTER messages PING messages that contains the variable payload generated by the random function above is send to the other agents which this agent wishes to communicate with.

```
SendSimMessage(new AgentMsg (MessageType.PING,
```

new MsgBall (getAPPid (),

currPosition,

currVelocity,

"PING"+payLoad+getAPPid ())),

timeIncrement,

subIMLPs).

SendSimMessage method interacts with the kernel and helps transmitting the messages between the application and kernel layers. The agents forward the messages to its respective IMLP that it subscribes to. The IMLP then forward the messages to after inspecting the destination of the messages to respective IMLPs to which the desired agents subscribes to which later delivers the messages to the respective agents. This was pretty much a peer-to-peer.

4.2 Fixed Clustering Approach

In fixed clustering scheme a fixed numbers of messages are clustered (of variable payload size). The configurable end –to –end delay parameter that buffers the messages before clustering start, defines the degree of aggregation. This algorithm assumes that message clustering always performs better than the unclustered approach and hence does not take other parameters like message size, network traffic etc into account before taking the decision to perform clustering. However, we later observed that under some scenarios like small message size clustering and unclustering had significant overhead rather than not aggregating messages. This overhead is

however addressed by our adaptive clustering mechanism that avoids clustering when it is not advantageous.

When a fixed number of messages are aggregated, the clustered payload is passed down to the SASSY kernel for transmission. The configurable delay parameter ensures that network units don't wait an indefinite amount of time before being sent. This value (Tfixed) in this scheme ensures that aggregation is performed, regardless of the number of message units, within some threshold.

Whenever IMLP receives a PING message from the agents subscribed to it buffers them in a hash map data structure for **clusterTime** of simulation time. This parameter increments by one each time when IMLP receives a PING message. The IMLP checks **clusterTime** against the threshold value before it transmits the clustered message. A code snippet describing the above mechanics is depicted below:

```
private void PING (MsgBall ball){
clusterTime++;
if (buffer.containsKey (ball.getAPPid())){
     buffer.get(ball.getAPPid()).add(ball.getMyMessage());
}
else {
     buffer.put(ball.getAPPid(),addMessage(ball.getMyMessage()));
     }
if(clusterTime > threshold){
sendSimMessage (new AgentMsg (MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString((IMLPright)));

sendSimMessage(new AgentMsg(MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString(IMLPacross));

sendSimMessage(new AgentMsg(MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString(IMLPbelow));

}
```

Figure 8: Fixed Clustering Approach Code Snippet

4.3 Adaptive Clustering Approach

As we discussed above when discussing the Fixed Clustering algorithm, clustering messages without o considering factors such network traffic and message size can adversely impact the performance of the simulation system. Hence we propose an algorithm that adapts to the prevailing network traffic dynamically within the SASSY kernel. This is similar in spirit to the

approach taken in AIDA (He et al. 2003) for sensor networks. Here the aim is to add an adaptive behavior that clusters when it is advantageous and avoids clustering otherwise.

To implement adaptive clustering we implemented message queues between the application layer and the kernel. All messages sent to the IMLP to be routed to respective agents are will have to pass through this message queue. As we discussed initially in this research each agent maintains its own state and there is no global view of the entire simulation system. The IMLP queue does not have to worry about the messages transmitted throughout the simulation system by all the agents.

Our adaptive scheme monitors the message queue to ensure that it contains at least one message waiting to be processed by the kernel. If so it infers that network traffic is high and this will lead to delay in the message transmission hence we perform clustering of messages until the threshold value. However, in scenarios where we observe that message queue is empty we transmit messages in their original form and perform no message clustering. This approach gave us a an adaptive clustering mechanism which make the clustering decision based on the internal network traffic. When this approach was compared with the fixed clustering algorithm approach we learned that for the same threshold value or the end-to-end delay the adaptive approach scores much better than the fixed one (see the experimental chapter). The code for the adaptive approach is depicted below:

```
private void PING(MsgBall ball){

      if(queue.size()!=0){

            if(buffer.containsKey(ball.getAPPid())){


buffer.get(ball.getAPPid()).add(ball.getMyMessage());

                  }

            else{

                     buffer.put(ball.getAPPid(),
addMessage(ball.getMyMessage()));

                 }

            queue.add(buffer);

              }

            else{

                  queue.add(ball);

                   }

            clusterTime++;

            if(clusterTime > 50){

                 if (communicationType.equals("relay"))

                        {
```

Figure 9 (a): Adaptive Clustering Approach Code Snippet

```
if(queue.peek().equals(buffer)){
   int  IMLPright=Integer.parseInt(getAPPid())+1;
   if(IMLPList.contains(IMLPright)){
      sendSimMessage(new AgentMsg(MessageType.CLUSTER,
       ball,buffer), timeIncrement, Integer.toString((IMLPright)));
    }
      int IMLPacross=Integer.parseInt(getAPPid())+noIMLPs;
      if(IMLPList.contains(IMLPacross)){
      sendSimMessage(new AgentMsg(MessageType.CLUSTER,
      ball,buffer),timeIncrement, Integer.toString(IMLPacross));
    }
    int IMLPbelow=IMLPacross+1;
    if(IMLPList.contains(IMLPbelow)){
    sendSimMessage(new AgentMsg(MessageType.CLUSTER,
    ball,buffer),timeIncrement, Integer.toString(IMLPbelow));
    }
    queue.remove(buffer);
}
else {
sendSimMessage (new AgentMsg (MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString((IMLPright)));

sendSimMessage(new AgentMsg(MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString(IMLPacross));

sendSimMessage(new AgentMsg(MessageType.CLUSTER,
ball,buffer),timeIncrement, Integer.toString(IMLPbelow));

queue.remove(ball);}}
```

Figure 9 (b): Adaptive Clustering Approach Code Snippet (contd.)

The figure below represents a high level overview of SASSY with the communication topology. It consists of both the unclustered and clustered approach discussed in this chapter.

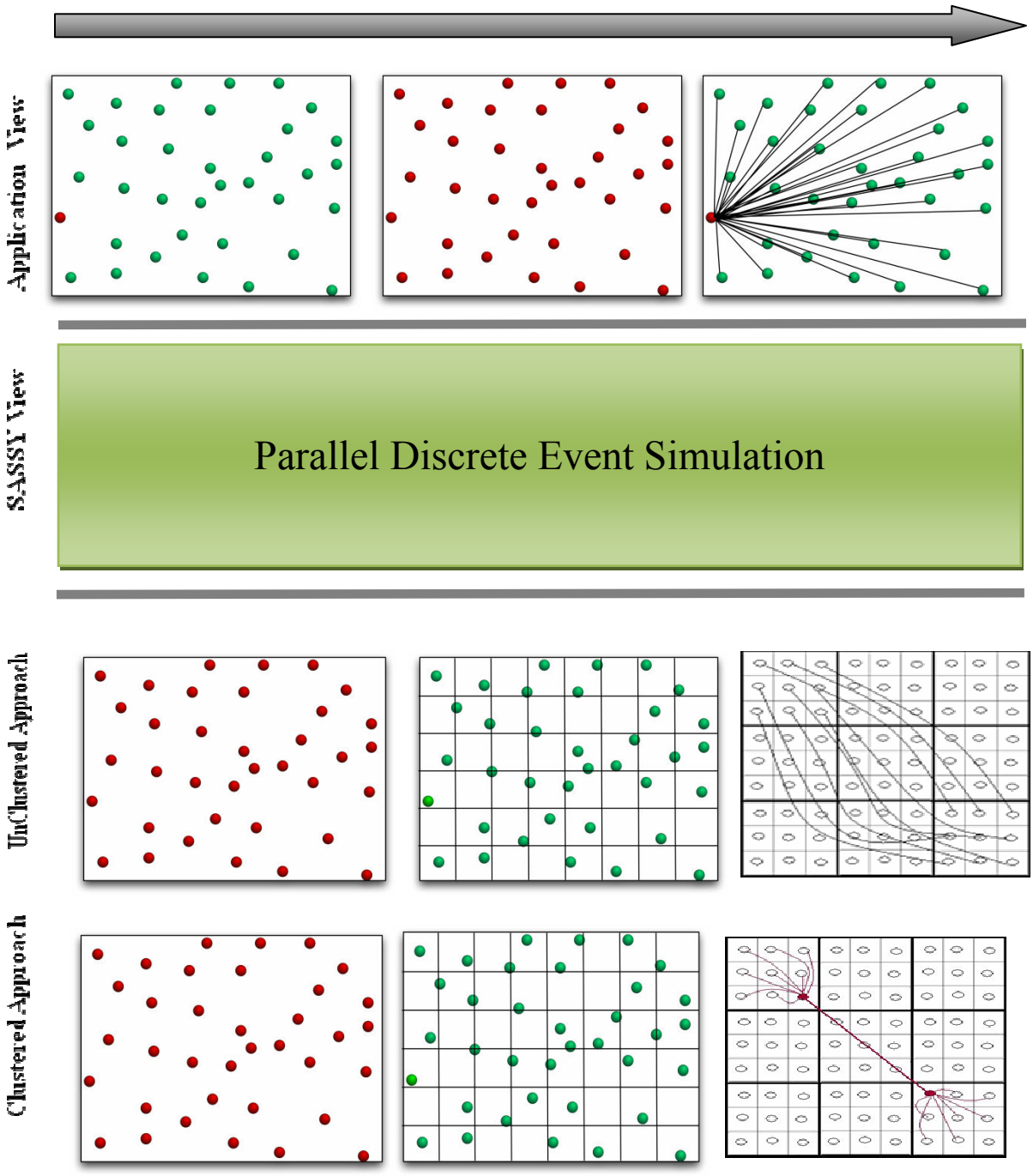

Figure 10: System Overview

CHAPTER 5

EXPERIMENTS

5.1 Experimental Platform

All experiments were conducted in a distributed environment with 100 IMLPs, 10 PE, 10- 300 agents and a payload size varying between 1 byte to 1KB for messages. Tests were executed on windows vista workstations, networked together with a gigabit Ethernet. Each workstation had a dual-core processor running at 2.6 GHz with 4 GB of RAM.

5.1.1 Performance Comparison for Agents with varying message sizes using Clustering and Unclustered Mechanisms

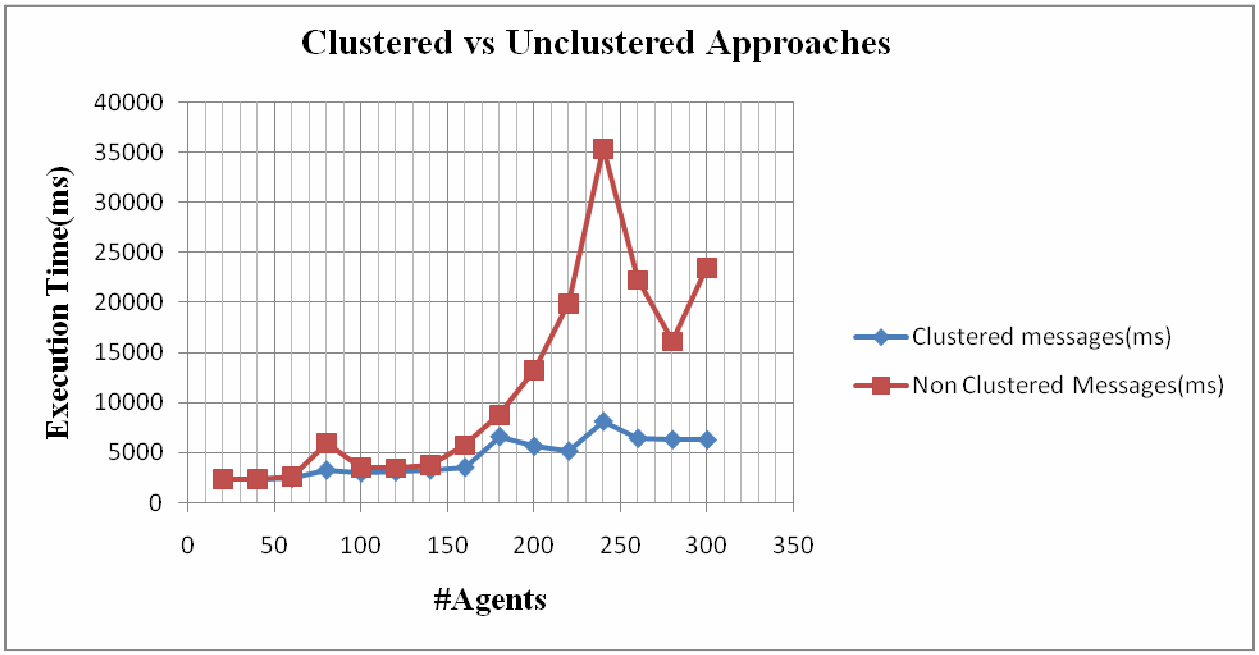

Figure 11: A Comparison of Clustered and Unclustered Message Approaches

In the first experiment we explored the utility of basic idea of message clustering in Agent Based Simulation systems and compared clustering with a system with no clustering approach on 10 machines (with PE per machine).

Quantitative results are shown in Figure 11. The X axis shows the number of agents and Y axis shows the execution time in milliseconds. Each data point is the average of 5 runs. Lower execution time indicates better performance. Initially clustering and non-clustering have about the same performance, but for the non-clustered case for runs of more than 150 agents we see a significant reduction in performance. We believe the peaks in the non-clustered plot are due to overloaded IMLPs, possibly because there may be a large number of agents subscribing to the same single cell creating a hot spot situation aggravating delays in service.

5.1.2 Distributed Execution Speedup for varying message sizes

We assessed how message clustering improved performance using a speedup metric. Speedup here refers to the ratio of the Unclustered and Clustered execution time for 100 IMLPs, 10 PE, a up to 300 agents and payload size ranging from 100 bytes to 1KB for messages. The aim of the experiment was to assess the performance gain of clustering over non-clustering using speedup, here speedup is defines as follows:

. **Speedup = Unclustered Execution Time (ms)/Clustered Execution Time (ms).**

Results are illustrated in Figure 12. In this plot each point represents the average of 5 experiments. The experimental set up was same as the previous experiment.

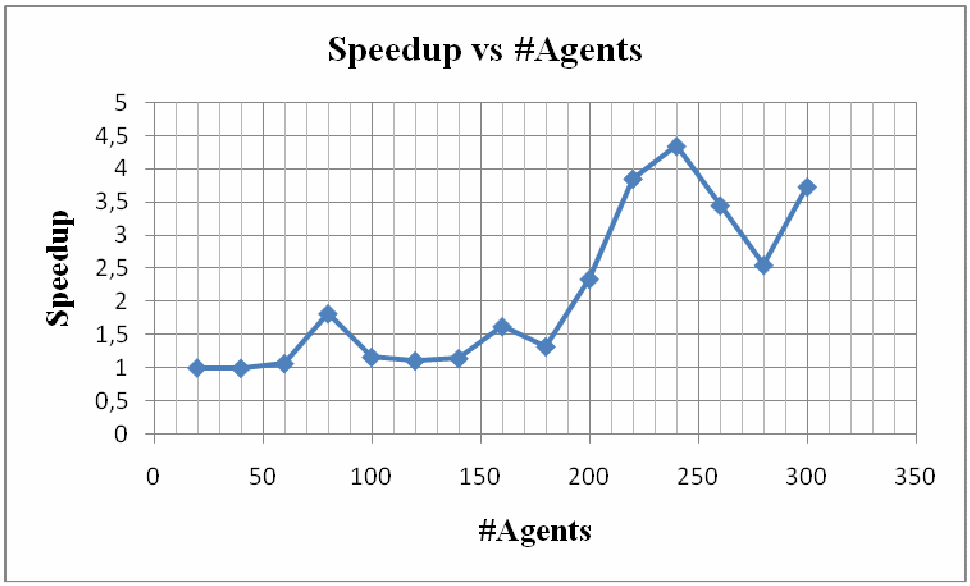

Figure 12: Speedup vs #Agents. Higher speedup values are better.

The large value of speedup indicates the amount of speedup gained as a result of the message clustering mechanism. In other words it indicates the performance gain achieved in terms of the execution time of the simulation with an increasing number of mobile agents. As already discussed in the previous experiment the peaks are due to the overloading of the IMLPs creating a hot spot that further aggravates service delays, e.g., there could occur a case wherein an IMLP is oversubscribed i.e. the number of agents subscribed to it may far exceed the capacity it can handle and can thus result in delays. With an increase in the number of active agents (transmitting messages of up to 1KB) the execution time for both clustered as well as unclustered approaches increases as expected. However, our results shows a speedup of either 1 or greater than 1 which indicates that clustering either performs as well or outperforms the non-clustering approach.

5.1.3 Data Model Representation For Message Clustering Systems

This experiment was conducted to assess the impact on communication overhead resulting from sending large packets (large message payloads). This experiment was conducted using 64 IMLPs spread across 8 PEs with 225 agents. To understand the message clustering impact we selected values that depended on the message(payload) size. The size of payload was not random and was fixed within each data model. Data models (X axis) 100 bytes, 200 bytes, 300 bytes up to 1,000 bytes for different message sizes depicted by a separate plot (100-900 bytes denoted by the legend in the plot).

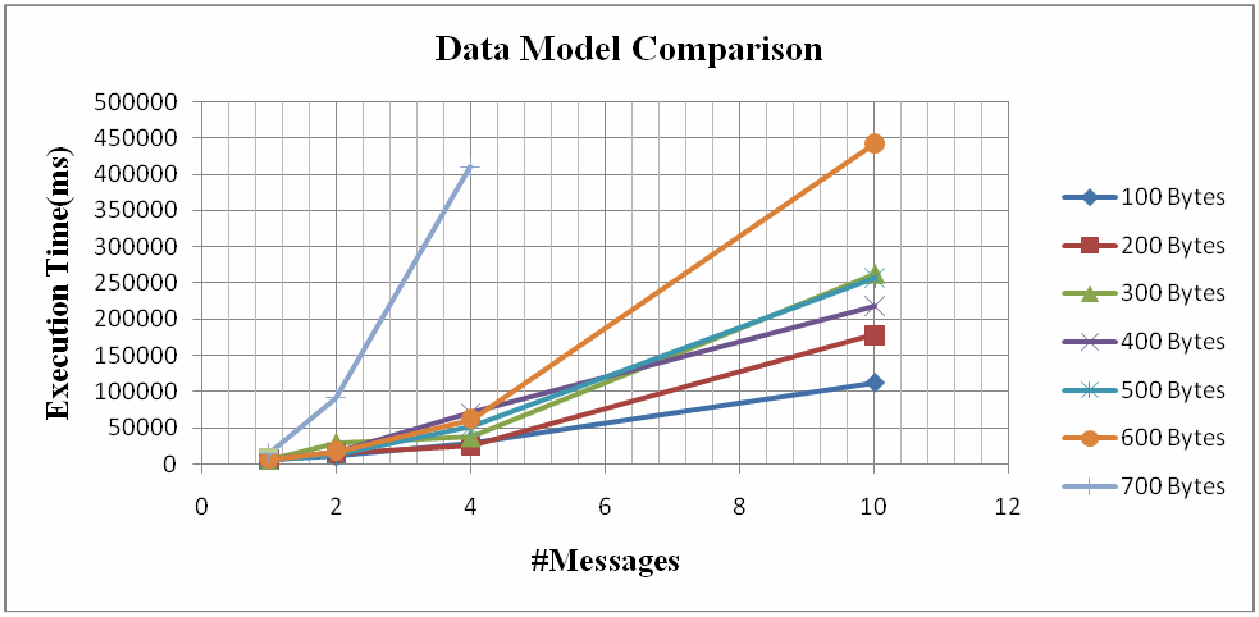

Figure 13: Data Model Comparison. Lower numbers indicate betterf performance

In each of these of the data models were sub broken into four groups of 2, 4 and 10 messages with each time the message payload or the data transmitted was kept constant.

E.g : Data Model : 100 bytes

No of messages	Size of each Message(bytes)
1	100
2	50
4	25
10	10

This was then compared with a single message transmission of the same size and their results were then compared. As expected it was found that in each data model the execution time for a single message rather than multiple messages was always less than those of 2, 4 or 10 grouped messages.

The message aggregation does induce message processing delays but as we observed in our first set of experiments that compared clustered and unclustered message processing techniques the delays due to message aggregation does not negatively impact the system performance compared to saturating the network with smaller messages. We tried the same experiment with data points higher than 1KB upto 2 KB with each data point disseminated into 1 (single message) of 1000 bytes, 10 messages (100 bytes) and accordingly for 100 and 1000 messages. The aim here was to conduct the same set of experiments for higher payloads. But unfortuntately this lead to java heap space errors so we kept message size 1 KB or less, but in future work we will investiage this limiation further.

In this plot we test the clustering of messages each time keeping the data point or message size constant. So if we have a message size of 100 we tested the clustering impact for 2 messages of 50 bytes each 4 messages of 25 bytes etc. So with increase in smaller messages execution time increases in clustering approach.

5.1.4 Performance measurement for 1KB to 10 KB message size

This experiment was conducted to assess the maximum size of the single message that is permissible to be transmitted in our aggregation framework before the Java Run time Environment (JRE) runs out of heap space memory. The experiment was conducted with 100 IMLPs on 10 PEs with 225 agents. The message size was kept constant at each data point with size increment of 1KB and each data point represents the average of 5 runs

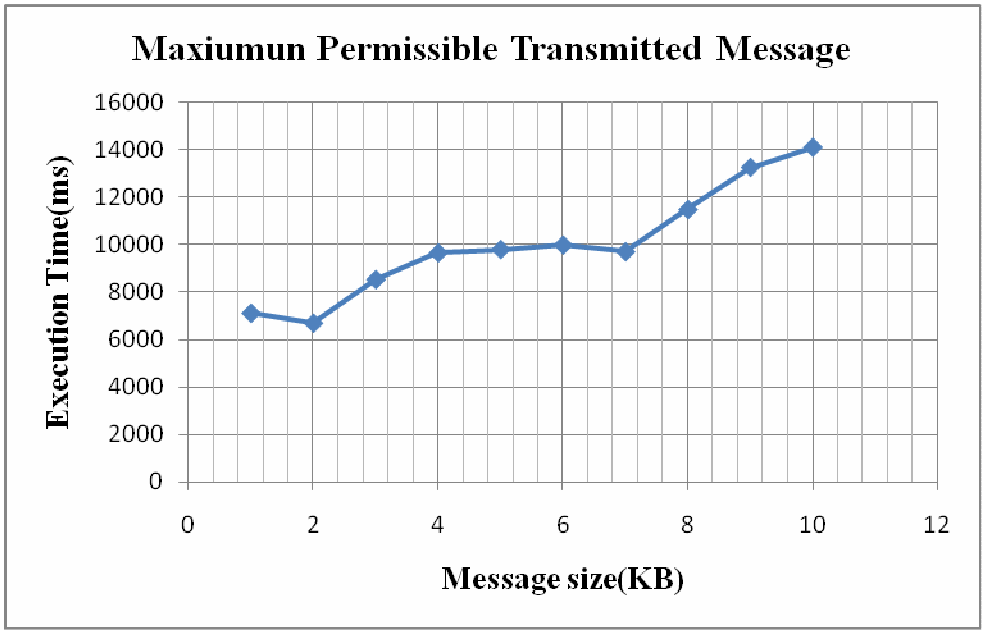

Figure 14: Message size (bytes) Vs Execution Time (ms). Smaller numbers are better

In the plot depicted in Figure 14 we observe that the execution time levels off between 4KB and 8KB (approximately) and then makes an increase in the execution for messages with higher payloads. This seems to indicate that processing time and in turn the aggregation of messages impacts the performance only when the payload increases by a threshold value (in our case 8KB).

5.1.5 Adaptive clustering approach comparison with fixed clustering approach

This experiment was conducted to test the performance of the simulation system with adaptive clustering compared to a fixed clustering approach. There are a number of possible factors impacting the performance of message transmission both internal and external: Internal factors include message processing time and message size, and external factors include latency, bandwidth and external work loads. To test the adaptive approach we varied the payload of messages to range between 1byte – 1KB. In this experiment the number of agents varied between 100 – 600 agents distributed over 4 PEs with 100 IMLPs, again each run (data point) represents the average of 5 runs.

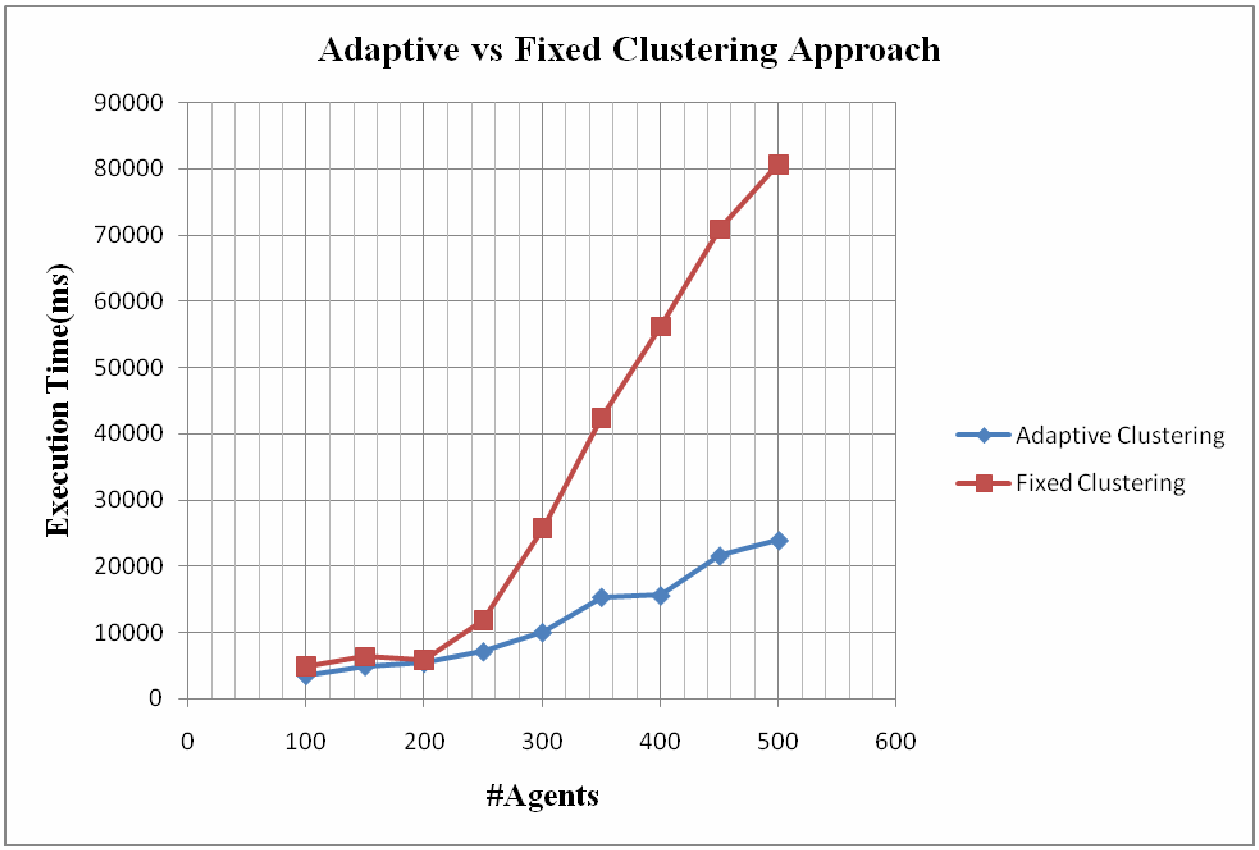

Figure 15: A Comparison of Adaptive and Fixed Clustering Approach. Smaller numbers are better

The results are shown in Figure 15; here we observe that at approximately 250 agents the plots starts to diverge. The execution time for fixed clustering increases at much higher rate compared to that of adaptive clustering. The adaptive clustering approach takes into account the internal traffic that depends on the processing time of the SASSY kernel and also the time taken to copy messages between network and local storage Clustering is performed when the queue that holds the unprocessed messages is full. Hence we observe that for the same amount of delay performing clustering of messages was not always helpful compared to the adaptive approach. The queue here will consist of unclustered as well as clustered messages

5.1.6 Advantages of Unclustering approach over Clustering Approach

The aim of this experiment is to highlight the scenarios wherein messaging clustering approach may not be advantageous as discussed in the previous experiment and give us an insight on how to parameterize 'adaptive' clustering, e.g., when the overhead of packing /unpacking does not offset the performance advantage. To study this we ran multiple iterations of fixed clustering with message sizes between 20 bytes -30 bytes with incremental sizes of 40 bytes, 50 bytes etc. The reason to have this small incremental change in the message size was to observe the impact of message clustering for messages of smaller sizes and note the threshold value at which clustering approach starts to have an impact on the performance of the system. The experiment was conducted on 4 PEs, 100 IMLPs and 300 agents.

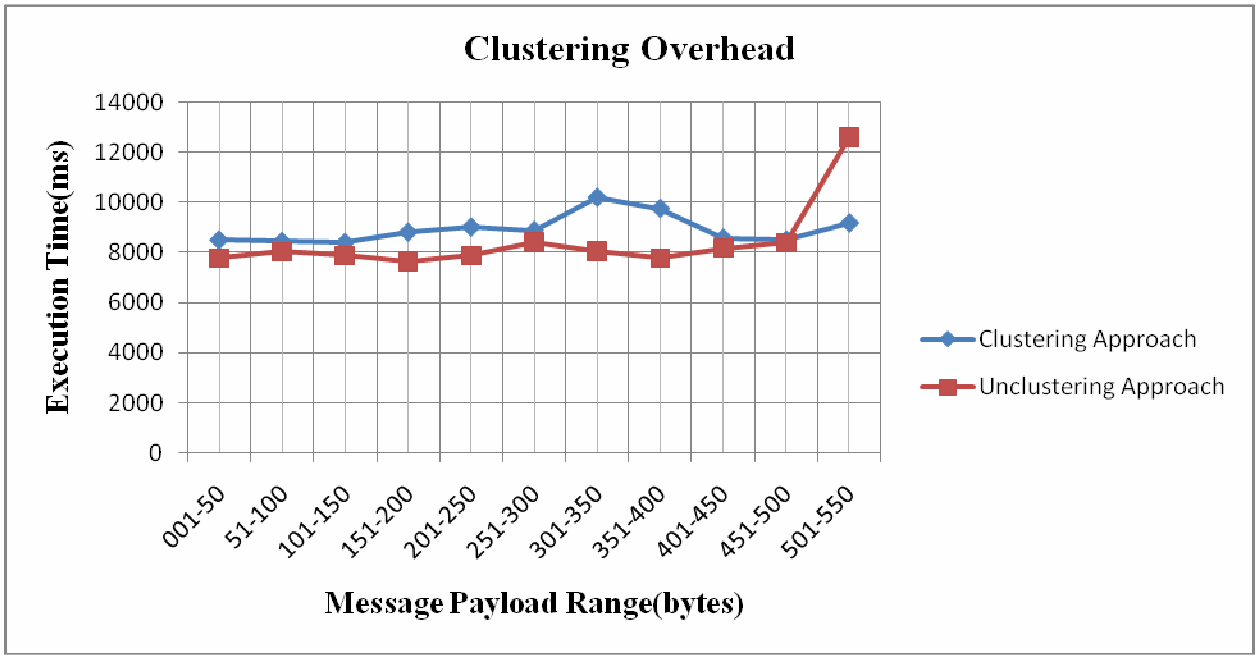

Figure 16: Advantage of Unclustering approach over Clustering Approach**.**

The X axis represents the Message payload range in bytes with increment of 10 bytes at each data point. The Y axis represents the execution time in milliseconds for the corresponding payload range on X axis.

The advantage of clustering was when the payload was 450 bytes or greater. We also observed that it was better not to cluster at smaller payloads i.e., the benefit did not overcome the overhead until a payload of 450 or greater.

From the experimental results discussed above we can infer that clustering performed with caution is a good idea for interactive agents in a complex agent based system. However, in order to obtain best results the decision to perform clustering should be done taking into the consideration the prevailing network traffic within the simulation system. In other words adaptive clustering can help improve the performance of a simulation system in a complex agent based system.

CHAPTER 6

CONCLUSION & FUTURE WORK

In this work we have shown that message clustering or "piggybacking" can improve performance in large scale distributed agent-based simulation. We demonstrated the potential benefit of message clustering in a Parallel Discrete Event Simulation (PDES) system. We evaluated clustering, non-clustering and adaptive clustering approaches by evaluating and implementing them with an agent-based java based discrete event simulation system called SASSY. SASSY is which is a hybrid large scale distributed agent-based simulation system that provides an agent–based API to a PDES kernel. The impact of message clustering on performance depends if we follow Fixed Clustering Approach or Adaptive Clustering Approach. Results indicated that the decision of message clustering should depend on message payload sizes and number of agents in the system. Performing clustering (as we called it as Fixed Clustering) may not be always beneficial like in case wherein the message payload size is small (1byte – 400 bytes) and the overhead of clustering and unclustering messages dominates. Hence we suggested an adaptive message clustering approach, which monitors when clustering is beneficial or not. We found that this approach always outperforms both fixed clustering and the unclustering approaches.

In this work the SASSY kernel API allows LPs to send serialized Java objects to each other for communication. For message transmission, SASSY currently uses Java's Remote Method Invocation (RMI) mechanism. In the future, we plan to replace this with TCP, achieving higher message throughout and lower latency. Moreover, to achieve a more realistic agent based scenario we would like the agents be moving across nodes (physical PEs). Currently, agents

move within a single PE. This would help us investigate communication patterns for a realistic and complex agent based system, mirrored in ants or bees simulation.

The movement of agents across nodes or processing elements (PE) can further help us distribute load of simulation system by aggregating the high interactive agents on a single node that will now communicate among themselves through shared memory instead of sockets. This would help us study the participation of external factors like load balancing techniques in Adaptive Message Clustering.

Also, in the current work the regions assigned to IMLPs are of uniform size. However, in practical this might not be the case with the size of regions dependent on the expected agents that region expects to hold.

REFERENCES

[1] Roy Friedman, Robbert Van Renesse. 1995 Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols.

[2] Toshihiro Takahashi Hideyuki Mizuta 2006 Efficient Agent – Based Simulation Framework for Multi- Node Super Computers

[3] Tian He, Brian M. Blum, John A. Stankovic and Tarek Abdelzaher 2003 AIDA: Adaptive Application Independent Data Aggregation in Wireless Sensor Networks.

[4] Tucker Balch, Adam Feldman, Andrew Guillory, Charles Isbell, Zia Khan, Stephen Pratt, Andrew Stein, and Hank Wilde 2005 How multi-robot systems research will accelerate our understanding of social animal behavior

[5] P Narsimhan, LE Moser, P.M. Melliar – Smith 1996 Message Packing as a Performance Enhancement Strategy with Application to the Totem Protocols.

[6] Sahar Adabiz, Sam Jabbehdari, Amirmasoud Rahmani, Sepideh Adabi 2008 A Novel Distributed Clustering Algorithm for Mobile Ad-hoc Networks

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

[8] 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.

[9] 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)

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

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

[12] 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).

[13] 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.

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

[15] 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)

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