Towards Simulating Billions of Agents in Thousands of Seconds

I.V. Aprameya Rao, Manish Jain, Kamalakar Karlapalem
Centre for Data Engineering
International Institute of Information Technology, Hyderabad, INDIA
{aprameya, manish_jain}@students.iiit.ac.in, kamal@iiit.ac.in

ABSTRACT
Building multi-agent systems that can scale up to very large number of agents is a challenging research problem. In this paper, we present Distributed Multi Agent System Framework (DMASF), a system which can simulate billions of agents in thousands of seconds. DMASF utilizes distributed computation to gain performance as well as a database to manage the agent and environment state. We briefly present the design and implementation of DMASF and present experimental results. DMASF is a generic and versatile tool that can be used for building massive multi agent system applications.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence, Intelligent Agents, Multi agent Systems

General Terms
Performance, Design, Management, Experimentation

Keywords
Multi Agent System, Simulation, Distributed Systems, Databases, Scalability

1. INTRODUCTION
Many Multi Agent Simulation Systems do not scale to a large number of agents. With dropping hardware costs, computer networks are present almost everywhere. They also do not utilize the advantages of distributing the simulation work across multiple computers in a networked environment.

The ability to pause and resume complex simulations is something that is missing in most Multi Agent System simulators. This applies more to simulators that use a main memory based simulation model (in which different threads are used for executing different agents).

Most systems come with their own interpreted language with steep learning curves. They are also not powerful enough for expressing common programming constructs.

The simulators that employ distributed computing are difficult to set up and maintain. There is no straightforward method of installing and deploying them. The time taken to build and deploy a simulation over a distributed system is considerably high. We have overcome these limitations in our system DMASF.

1.1 Design features of DMASF
An agent in DMASF is represented by an agent type and an agent id. Each agent type can have an independent set of properties. Multiple agent types and multiple agents of a single type are permitted. An agent cannot change its type directly. To implement type changing, the environment would need to kill this agent and create an agent of the new type. In DMASF an agent lives until it is explicitly killed by the environment or the simulation ends. A host is an instance of DMASF running on a computer. A computer can run multiple hosts at the same time. Our model has a set of simulators on each host. These simulators are run in separate threads and are responsible for executing the code for sets of agents.

We use a database for storing agent state information. Databases already have excellent query mechanisms and are very robust. Databases can easily store and retrieve information for billions of tuples and thus help in achieving impressive scale ups. In DMASF, a fixed number of agents are kept in main memory at a time while the rest are flushed to a database system so that retrieval and update is performed efficiently. We need to provide the same view of the world at each iteration to all the agents. We cannot commit the updates made by an agent to the database immediately as the simulation would then present two world views. Thus, we have to wait until all agents have finished updating. Again, due to the sheer number of such updates we cannot store these updates in main memory. Hence, we keep a fixed number of updates in main memory and write the other updates to secondary storage. DMASF has a setting to override this default behavior if the simulation requires updates to be visible immediately.

Another challenge in implementing a distributed computational system is to schedule or decide which agents should be simulated on which machines. We cannot allocate an equal number of agents to each host as slower hosts will then tend to slow down the faster ones. Therefore, DMASF has a dynamic scheduler that assesses the performance of each machine and dynamically schedules and load balances agents on them.

The hosts in DMASF are organized in client-server architecture. The server decides which agents are to be

AAMAS’07, May 14–18, 2007, Honolulu, Hawai'i, USA.
Copyright 2007 IFAAMAS.
simulated on which host. It is also responsible for the synchronization among hosts.

Query results that give common world states are cached so that the simulation runs faster. In a simulation in which agents had to move in a circle (refer section 2.1.2), this caching reduced simulation time from $O(n^2)$ to $O(n)$ where $n$ is the number of agents. Whenever an agent requests such information, it is provided from the cache instead of running the query. This reduces the number of database queries, the load on the database system, and avoids redundant computation.

The contribution of this paper is to build a new system with the following design goals: (1) A fast lightweight core, (2) Distributed computing for high performance, (3) Scalability for hundreds of millions of agents without the GUI, (4) A separate extensible GUI, (5) Saving the simulation state so that it could be resumed at any time, (6) Easy extensibility.

The details of implementation of the DMASF framework are not presented here but are available as technical report and downloadable software [10].

2. EMPIRICAL VALIDATION

Due to the lack of space here, we do not show details of all the experiments performed. We concentrate on results that demonstrated the scalability as well as stability of DMASF.

2.1 Sample Worlds and Agents

2.1.1 K-P Simulation

One of the simulations created was the K-P benchmark, where $K$ refers to the number of agents in a group, $P$ is the number of messages sent by each agent on receiving $P$ messages from the other agents in the group. This simulation was designed to test the efficiency of messaging.

The simulation was run with 10000 agents and two hosts. The graph of Time taken v/s $P$ (the number of messages sent in a group) is shown in Figure 1.

Since messages are being stored as tuples in a table, we expected a linear increase in the time with a linear increase in the number of messages in the system. It can be observed that as $P$ increased, the number of messages in an agent group increases. Therefore with linear increase in the number of messages the simulation time also increased linearly. For the same $P$, with a larger $K$ (group size) there are fewer total messages in the system and hence the time taken is less.

2.1.2 Circle Simulation

The circle simulation environment is where the agents are spawned in a random manner in a 2D plane. This is an extension of one of the examples provided with NetLogo [5]. The agents can only see other agents. The agents must move around in a circle whose center depends on the location of the other agents. This simulation was run on two different setups. The first setup involved two hosts running on a machine with four 3GHz processors and 2GB of main memory. We gradually increased the number of agents to see how well the simulation would scale. The simulation scaled linearly (refer Figure 2). In both the setups the MySQL database was on this machine. The second setup involved running the simulation with one million agents. We increased the number of hosts and plotted the resultant gain in performance (refer to Figure 3). The machines that were used for simulation were standard desktop computers with AMD 1.6GHz processors and 512 MB of main memory over a 100Mbps Ethernet.
noticed that the performance gain on going from one to two 
hosts is much more than the performance gain on going from 
ten to eleven hosts. With one host, we are simulating 
1,000,000 agents on a single machine. With two hosts this 
becomes 500,000 on two machines providing a significant 
 improvement. However, when we have 10 hosts, we are 
processing 100,000 agents on each host. When we increase 
this to 11 hosts, we are processing approximately 90910 agents 
on a host. Thus, the performance gain in the latter case is 
significantly less than the performance gain in the former.

We further built a FireFighter simulation modeled along the 
lines of the RoboRescue [8] competition. It had three kinds of 
agents: Helicopter, Rescue Vehicle and Smoke (refer to Figure 
4). The environment would randomly spawn smoke agents. 
The helicopter would be scanning the world looking for new 
smokes. On finding a new smoke (fire), it would alert the 
rescue vehicle agents. Each rescue vehicle agent would go 
towards the smoke closest to it and try extinguishing the fire. 
The smoke agents would have a natural lifetime that 
corresponds to the time in which the fire would burn out.

![FireFighter simulation](image)

**Figure 4. FireFighter simulation**

3. SUMMARY

A lot of work has been done in the field of developing Multi 
Agent System Simulators. SPADES [1], MASON [3], 
NetLogo and JADE [6] are some of the more popular 
simulation toolkits. JADE (Java Agent Development 
Environment) which is a middleware for developing and 
deploying Multi Agent System Applications. SPADES is also 
a Distributed Multi Agent Simulation Environment which is 
not language specific and allows agents to be written in any 
programming language. The agent code interacts with 
SPADES over UNIX pipes. MASON is a light, fast, scalable 
discrete-event Multi Agent simulation library written in Java. 
It also has a separate visualization layer for viewing 
simulations. NetLogo is a desktop simulation toolkit that 
 scales well for small number of agents. It uses its own 
interpreted language for writing agent simulations.

Multi Agent System technology can be used to simulate 
complex environments at both microscopic and macroscopic 
levels. Therefore, it is required to have a simulation toolkit to 
cater to both these needs. Many of the current Multi Agent 
Simulation toolkits either cater to microscopic simulation for 
very small environments (~10,000 agents) or do only 
macroscopic simulations. One of the challenges taken up in 
this paper is to provide a generic toolkit that can simulate a 
very large number of agents in a relatively short time by using 
distributed computing. Our results show that we can 
potentially simulate one billion agents in around 250,000 
seconds. With increased number of systems used for 
simulation it is feasible to bring down this time further.

One major advantage of our simulation toolkit is that it can be 
used to rapidly implement and deploy small Multi Agent 
System driven simulations. Thus, it is amenable for use in 
Multi Agent System course projects. Since DMASF is built in 
Python it is easy to plug-in existing MAS decision modeling 
systems such as MDP into DMASF. One of the limitations of 
our system is the computational mismatch between the GUI to 
show the simulation results and the backend that actually does 
the simulation. With advances in GUI rendering and related 
technologies it would be feasible to have real-time observation 
or animation of a very complex Multi Agent Simulation with 
one billion agents. If such GUI technology is not available 
then other appropriate solutions are needed for handling this 
mismatch. We are currently working on simulation of very 
complex environments using this toolkit.

REFERENCES

[1] Patrick F. Riley, and George F. Riley. *Spades – A 
Distributed Agent Simulation Environment with 
Software-in-the-Loop Execution*. Proceedings of 

[2] Luis Mulet, Jose M. Such, and Juan M. Alberola. 
*Performance Evaluation of Open-Source 
Multitagent Platforms*. International Conference on 

and Keith Sullivan. *MASON – A New Multi-Agent 
Simulation Toolkit*. Society for Computer 

Karlapalem. *System Issues in Crowd Simulation 
using Massively Multi-Agent Systems*. Workshop on 

[5] Seth Tissue, and Uri Wilensky. *Netlogo: Design and 
Implementation of a Multi-Agent Modelling 
Environment*. Agent2004 Conference.

Rimassa. *JADE – A FIPA compliant agent 
Framework*. Proceedings of The Practical 
Applications of Intelligent Agents and Multi-Agent 
Technologies (PAAM), 1999.

Managing Millions of Agents*. Proceedings of 


Intelligence: A Modern Approach*, Prentice Hall 

[10] DMASF Homepage: [http://cde.iit.ac.in/~dmasf/](http://cde.iit.ac.in/~dmasf/)