IMPROVING THE PERFORMANCE OF DISTRIBUTED MULTI-AGENT BASED SIMULATION

Dawit Mengistu

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2011:04

School of Computing

IMPROVING THE PERFORMANCE OF DISTRIBUTED MULTI-AGENT BASED SIMULATION

Dawit Mengistu

Blekinge Institute of Technology Doctoral Dissertation Series
No 2011:04

IMPROVING THE PERFORMANCE OF DISTRIBUTED MULTI-AGENT BASED SIMULATION

Dawit Mengistu

School of Computing
Blekinge Institute of Technology
SWEDEN

© 2011 Dawit Mengistu
School of Computing
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2011
ISBN: 978-91-7295-198-3
Blekinge Institute of Technology Doctoral Dissertation Series
ISSN 1653-2090
urn:nbn:se:bth-00485

IMPROVING THE PERFORMANCE OF

DISTRIBUTED MULTI-AGENT BASED SIMULATION

Dawit Mengistu

Abstract

This research investigates approaches to improve the performance of multi-agent based simulation (MABS) applications executed in distributed computing environments. MABS is a type of micro-level simulation used to study dynamic systems consisting of interacting entities, and in some cases, the number of the simulated entities can be very large. Most of the existing publicly available MABS tools are single-threaded desktop applications that are not suited for distributed execution. For this reason, general-purpose multi-agent platforms with multi-threading support are sometimes used for deploying MABS on distributed resources. However, these platforms do not scale well for large simulations due to huge communication overheads. In this research, different strategies to deploy large scale MABS in distributed environments are explored, e.g., tuning existing multi-agent platforms, porting single-threaded MABS tools to distributed environment, and implementing a service oriented architecture (SOA) deployment model.

Although the factors affecting the performance of distributed applications are well known, the relative significance of the factors is dependent on the architecture of the application and the behaviour of the execution environment. We developed mathematical performance models to understand the influence of these factors and, to analyze the execution characteristics of MABS. These performance models are then used to formulate algorithms for resource management and application tuning decisions.

The most important performance improvement solutions achieved in this thesis include: predictive estimation of optimal resource requirements, heuristics for generation of agent reallocation to reduce communication overhead and, an optimistic synchronization algorithm to minimize time management overhead. Additional application tuning techniques such as agent directory caching and message aggregations for fine-grained simulations are also proposed. These solutions were experimentally validated in different types of distributed computing environments.

Another contribution of this research is that all improvement measures proposed in this work are implemented on the application level. It is often the case that the improvement measures should not affect the configuration of the computing and communication resources on which the application runs. Such application level optimizations are useful for application developers and users who have limited access to remote resources and lack authorization to carry out resource level optimizations.

Acknowledgements

I would like to take this opportunity to express my indebtedness to all who supported me in this PhD thesis. Because it is difficult to list the names of all of them here, I would mention only the following people for their special contribution in my work.

First, I owe my deepest gratitude to my advisors Professor Paul Davidsson and Professor Lars Lundberg whose thoughtful ideas and critical reviews helped me to develop the research. Without their guidance and kind support, the thesis would not have come this far.

I am sincerely grateful to Dr. Peter Tröger, for his insightful advice and sharing his knowledge in distributed systems at a critical time in the course of this work. I also appreciate the helpful discussions and moral support I received from my colleagues at BTH. I am also indebted to Hasso-Plattner Institute (HPI) of Germany for giving me the opportunity to conduct a part of my research at their facilities. The discussions I had with researchers there were highly valuable.

Finally, I would like to express my gratitude to my family for their support, encouragement, understanding and patience during the challenging times of my work. Thank you all for being with me throughout!

List of Publications

The thesis is based on the following publications:

1. Davidsson, P., Holmgren, J., Kyhlbäck, H., Mengistu, D., Persson, M. *"Applications of Multi-agent Based Simulation"* Multi-Agent-Based Simulation VII, Lecture Notes in Computer Science, Vol. 4442, Pages 15-27, Springer, 2007.

2. Mengistu, D., Davidsson, P., Lundberg, L. *"Middleware Support for Performance Improvement of MABS Applications in the Grid Environment"* Multi Agent Based Simulation VIII, Lecture Notes in Computer Science, Vol. 5003, Pages 20-35, Springer, 2008.

3. Mengistu, D., Tröger, P. *"Performance Optimization of MABS Applications on the Grid"* Proc. Eighth IEEE International Symposium on Cluster Computing and the Grid, Pages 560-565, France 2008.

4. Mengistu, D., Tröger, P., Lundberg, L., Davidsson, P. *"Scalability in Distributed Multi-Agent Based Simulations: The JADE Case"* Proc. Second International Conference on Future Generation Communication and Networking Symposia, Vol. 5, Pages 93-99, China 2008.

5. Mengistu, D., Davidsson, P., Tröger, P., Lundberg, L. *"Performance Modeling and Optimization of Agent Based Simulations in Distributed Environment"* Proc. The Social Simulation Workshop at the International Joint Conference on Artificial Intelligence (SS@IJCAI) pages 79-92, USA 2009.

6. Mengistu, D., Löwis, M.v. *"An Algorithm for Optimistic Distributed Simulations"* Proc. IASTED-ACM conference on Modeling, Simulation and Identification (MSI-2009), China 2009.

7. Mengistu, D., Davidsson, P., Lundberg, L. *"Approaches to Distributed Multi-Agent Based Simulation"* Submitted for journal publication, 2010.

8. Mengistu, D., Davidsson, P., Lundberg, L. Tröger, T. *"Migration Heuristics and Performance Models for Distributed Multi-Agent Based Simulations*" Submitted for journal publication, 2010.

Contents

ONE

I. INTRODUCTION

Major advances have been witnessed in the field of distributed computing over the last several years. The infrastructure of distributed systems has improved by orders of magnitude in terms of processing power and data communication speed, and is becoming more affordable at the same time. These developments motivated the emergence of new types of distributed applications that were never thought of, or were once considered to be intractable.

Along with this progress, new computing paradigms and technologies were conceived to address the specific requirements of distributed applications. The Web, electronic commerce, e-learning, and other application areas flourished as a consequence. Many applications run satisfactorily on distributed infrastructure as the architecture of these applications was factored into the design and development of the infrastructure, the programming and execution models. However, some applications were rather after-thought, i.e., they were conceived as spin-off benefits of the new computing paradigm. Other application areas had yet their roots in a single processor execution model and needed to be redesigned to fit into the established distributed computing paradigms.

Scalability is a key requirement in distributed systems. Applications need to be designed to utilize the distributed infrastructure efficiently and fairly. The development of distributed applications demands meeting this requirement, without violating another requirement: the correctness of the application execution and its results. These two requirements are often conflicting and pose a serious challenge to designers of distributed applications. The problem to be handled in this situation is essentially that of performance optimization of the distributed application.

The purpose of this research is to study performance improvement strategies for an application domain that has recently gained wide attention, namely distributed multi agent based simulations (MABS). MABS is a modern simulation approach based on micro modelling as opposed to the traditional equation-based (macro) modelling. It is a promising approach to

the simulation of systems containing a large number of interacting and/or heterogeneous entities. This simulation technique has been received with great interest by researchers in diverse fields, such as the social sciences. However, MABS is not a friendly application in today's distributed execution environment due to its inherent architectural features that are sources of serious performance bottlenecks. This research aims at the tasks of investigating these sources of performance problems and finding ways to improve the execution characteristics of distributed MABS applications by tackling these problems. This chapter gives a brief review of distributed computing and background information on MABS followed by the research questions dealt with in this work and the methodology applied. Next, the main contributions of this research and its limitations are discussed. Finally, concluding remarks and directions for future work will be presented.

1.1 Distributed Computing

The concept of distributed computing has been around since the birth of data communication networks. An accepted description of a distributed system is that it is an interconnection of computing resources spread geographically, but utilized as a unified resource to solve a certain task. The seemingly unending growth in processing power and interconnection bandwidth has led to new computing paradigms [3].

Several reasons are often cited for using distributed systems. The first is that in certain applications, the processed data is acquired, stored and consumed at computers in different physical locations and thus such systems are inherently distributed. The web and the Internet are the best examples of this category of applications. The second reason is that a single computer system is less reliable as it can be a single point of failure while a distributed system can be fault tolerant and dependable. In some application areas, loss of data or interruption of service due to computer downtime is detrimental. The reason which is most relevant to this thesis is: some applications demand a large amount of computing power that is not usually available from a single source. In such cases, it is not practical to run the applications on a single computer and get the result in a reasonable time. The issue of how best to deploy and execute an application over distributed resources in an efficient

matter draws interest among computer performance researchers and application developers.

Although there are significant differences in the types of distributed applications, certain key requirements are commonly needed in all application types from the perspective of reliability and usability [9]. These requirements can be grouped into:

Availability and fault tolerance requirements: Availability refers to the probability that a distributed system is operational and running at any given time. It is a concept commonly used in servers and telecommunication systems which are expected to be functional all the time, if possible. Fault tolerance is the ability of a system to restore operations or recover from failure and continue providing services even when some of its components are down.

Consistency requirements: A distributed system is expected to maintain coherent information and ensure integrity of data over all of its processing units. Managing concurrency and synchronization of operations in a distributed system is part of the challenges to meet the consistency requirements.

Security requirements: Some applications and resources need to be protected against unauthorized use. Authentication mechanisms, establishing trust among the different players in a distributed computing environment is not a trivial task.

Transparency requirements: As explained earlier, a distributed system should appear as a single resource to its users who need not know the type of hardware or software deployed at each physical location. This is commonly achieved by middleware, a software layer running on top of operating systems. This layer abstracts away all intricacies of the underlying layers and offers a single view of the system to the user, no matter what the type of the resource an application is running on. As a result of this illusion of a simple and single system, software designers and architects are often tempted to take inaccurate assumptions about the execution characteristics of their applications [4].

Performance and scalability requirements: The scalability aspect deals with the ability to deploy an application on a wide range of computing resources such that the application obtains a desired throughput and quality of service regardless of variations in usage configurations. The performance

aspect considers the predictability of an application's response time which is related to its efficiency in utilization of resources such as processing capacity, memory, network bandwidth, etc. As explained earlier, this requirement is the focus of this research and will be dealt with in detail in this thesis.

In order to meet the above requirements, several hardware configuration architectures, programming models, operating system enhancements and middleware stacks emerged leading to different computing paradigms. Some of these paradigms came about as a result of evolving service provisioning requirements, new business models, and growing affordability of large processing capacity. It is difficult to agree on a unique classification of the paradigms as it depends on the aspect of interest one has. However, for the purposes of this work, we can use a classification based on the hardware and the infrastructure configuration. Because the programming and deployment models are often dependent on these configurations, it is reasonable to use this classification for the type of performance studies considered in this thesis.

Cluster: In this configuration, a group of usually inexpensive computers called nodes are connected in a high-speed local area network. In a typical configuration, users can get access the cluster through one of the nodes called the head node only. Clusters are designed with a view to satisfying the availability or performance requirements of applications they are intended to host. High availability clusters are used to improve service availability by employing fault tolerance and redundancy techniques. On the other hand, in high performance clusters, an application task is split into smaller chunks which can be executed on each node simultaneously and independently. Parallel programming models such as MPI are used on clusters for partitioning and allocating tasks on the nodes, facilitating inter task communication, managing concurrency and collecting the end results.

Grid: Heterogeneous computer resources spanning over geographically distributed administrative domains are loosely coupled to form a computational Grid that provides an aggregate computing power to the users [1]. The resources can be clusters, super computers, desktop machines or mainframes. Due to the heterogeneity of the resources and their geographical separation, most requirements of distributed systems explained earlier are harder to meet in the Grid. For this reason, it is not suited for interactive and tightly coupled applications.

Of primary importance in evaluating an application's feasibility for the Grid is its communication aspect. If the tasks allocated to worker nodes need

to communicate or synchronize with each other very often, then that application is regarded as an unsuitable candidate for the Grid. Search applications are the most successful in the Grid environment as there is virtually no need for communication between tasks deployed at individual workers. In such applications, the most challenging step is splitting up the work in to smaller and balanced tasks manageable by worker nodes.

Cloud: With the maturity of virtualization technologies and growing demands for better service provisioning models, the cloud computing paradigm emerged. This paradigm applies new technical and business models to address the computing needs and the service delivery aspects. On the technical side, virtualization allows multi-tenancy, i.e., several virtual hosts can be run on one physical host, each host appearing to be running on a separate dedicated machine. The processing units, the memory and storage on a server are partitioned and several instances of virtual hosts that may run a variety of operating systems configured in different ways are created in these partitions. In most cases, because virtual hosts can be on the same physical machine, the network latency is lower for distributed applications deployed on these hosts.

In this research, the performance characteristics of agent based simulation applications are studied in the above computing environments. Because these environments have inherent differences, it would be appropriate to study in which way applications have to be designed and developed for each of these cases.

1.1.1. Performance Modelling and Prediction

Applications do not always run efficiently in all types of distributed execution environments. In other words, it is difficult to get the theoretically achievable throughput of computing resources at all times. Several reasons can be attributed to this problem, such as network congestion, node failures, uneven allocation of tasks to worker nodes, etc. Performance analysis in computer systems is used to evaluate the extent to which a system achieves its pertinent objectives, such as rate of work output, utilization of devices and satisfaction of environmental constraints. The analysis of computer performance is useful not only for evaluating effectiveness, but also for identifying performance problems and bottlenecks [2].

The implementation of performance analysis techniques in distributed systems is complex due to several factors such as resource heterogeneity, availability, load characteristics, etc. If the execution environment is robust and responsive enough, it can take necessary measures such as redistributing of tasks among compute resources, rescheduling the tasks for a later time or for different resources. The types of performance problems in distributed systems can be grouped in to two:

Resource (infrastructure) related: In this case, the problems are related to the environment itself. Examples include poor scheduling strategies, communication problems, other resource problems such as memory management, cache management, storage allocation, etc. These problems can be overcome by adopting efficient resource management strategies. For this purpose, system administrators often employ resource monitoring tools to monitor problematic areas.

Application related: These types of problems arise when an application is not designed to match its execution environment or deployed on the wrong system. For example, an application that was designed to work efficiently on a symmetric multiprocessor would likely perform poorly on the Grid. The main source of performance loss in this case is mostly inherent to the architecture of the application itself and execution efficiency and speed up cannot be achieved by increasing the number of computer resources.

Performance modelling is essential to build mathematical prediction models that can be used to understand the execution behaviour of applications and predict their performance in a distributed environment. These models have two important goals. First, they help application architects to design software that executes efficiently on a given set of resources by improving concurrency and communication related operations. Second, they help system engineers to devise strategies that can be used to tune applications on the fly, to apply more robust resource management such as dynamic load balancing, task reallocation and migration. For this reason, performance modelling and prediction modules are often implemented as components of resource management programs in high performance computing sites.

Because all distributed applications are not alike in their execution behaviour, it is difficult to generalize their performance characteristics. Some applications are computation or memory intensive, while others have minimal computation tasks interleaved with frequent communication tasks. Certain applications are massively multi-threaded, others could be executed as

workflows. Because of their architectural and implementation diversity, it is not possible to build a uniform performance model that can fit to all types of applications.

One feasible approach to deal with performance issues would be to categorize applications into domains of similarity in terms of their execution behaviour. Performance prediction models are therefore built for a given application domain by making certain valid assumptions and approximations justified by communalities among applications in that domain. The accompanying resource management system should accordingly consider these differences. In this thesis, one of the intended tasks is to investigate the usage of a domain-specific resource management module for MABS. The task involves studying their architecture, deployment and execution characteristics, and building performance prediction models. In the next section, a brief description of agent based simulation and the technical aspects of its implementation will be given.

1.1.2. Performance Monitoring Tools

A generic performance model of an application can serve as indicator of the application's execution behaviour and resource requirements. It can give an insight of the application, whether it is memory, CPU, or network intensive. The model gives a gross estimate of execution time or resource utilization under ideal or no load conditions. For the model to be fully usable in a diverse range of load conditions and resource variations, predictions have to be updated dynamically in real-time.

Performance prediction models can be integrated with resource monitoring tools to meet the aforementioned aim in high performance computing systems. In several implementations, the configuration of performance monitoring and prediction modules follows a control theoretic approach. Resource monitoring data is used to update prediction models, the predictor output is then used by resource management systems to affect scheduling and allocation decisions. The outcome of the decisions is then feedback through the monitoring system to the predictor and this process continues in a cyclic manner. In the Grid environment, a resource discovery service maintains a directory of the Grid's resources by consuming the output of the monitoring system. The directory contains a collection of information on availability and capacity of the resources at the individual sites of the Grid.

The Global Grid Form (GGF) Performance Working Group proposed a generic resource monitoring architecture called the Grid Monitoring Architecture (GMA) [1]. In this architecture, the components of the resource monitoring system are divided into three layers. The bottom layer is tasked with collecting performance data from the Grid hosts, the connection network and running applications. The top layer deals with performance analysis and prediction while the middle layer handles management and presentation functions. The figure below shows a high level specification of this architecture.

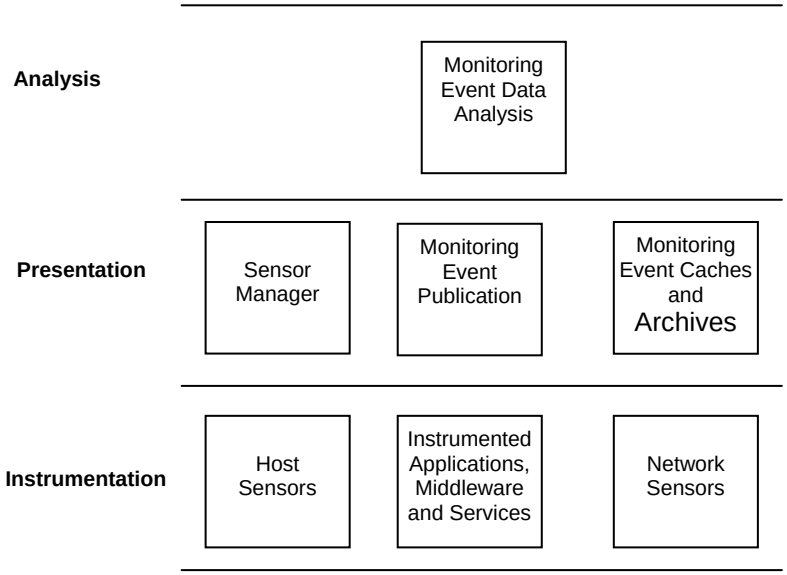


Fig. 1. A generic performance monitoring architecture. [1]

Different performance monitoring tools are used in distributed computing environments [6]. The difference in the tools is explained in the following ways:

1. whether the monitoring is performed on-line or off-line,
2. whether the tool has API support to be directly integrated with user applications, and
3. the specifics of the tool with respect to the components of the resource (CPU, memory, network, etc).

The Ganglia monitoring tool is deployed in the test bed used for the experimental work of this research. It is a hierarchically designed distributed monitoring tool for high performance computing systems. It has evolved into a planetary-scale Grid infrastructure monitoring tool from a cluster level product. The Ganglia tool employs two types of daemons. The first daemon is used to collect monitoring data at individual node (host) level. The second daemon collects node level measurements and aggregates them to produce site level monitoring data. Communications between the daemons occur through a multi-cast based listen/announce protocol to monitor states within a cluster and heartbeats from nodes to determine their availability and maintain membership information. Each node within a cluster monitors its own resources based on built-in performance metrics and sends multicast packets with monitoring data to a well known multicast address [7]. Each node maintains monitoring data of all other nodes in a cluster. The monitoring data collected this way can be used at the upper layers for resource discovery and performance prediction purposes.

1.2. Multi Agent Based Simulation

Along with the exponential growth of computing power over the last decades, important developments and methodological shifts have taken place in scientific research. The field of simulation is a living example of this phenomenon, where micro simulation models are increasingly replacing the long-established macro simulation techniques. The availability of affordable computing power encouraged simulation researchers to focus on individual-based micro modelling, instead of aggregate mathematical equations to represent system dynamics at macro level. The field of artificial intelligence (AI), which was left in the back burner for some time, become of a growing interest, shored up by availability of adequate processing power and the emergence of new parallel processing and distributed computing methods and technologies.

The revival of AI in the 80's and early 90's led to the appearance of intelligent agents in the computing arena. In broadly terms, an agent is a system with defined goals and objectives that can perceive what is happening in its environment. It would then makes autonomous decisions (e.g., according to a set of rules) based on what it perceives and accumulated

knowledge (beliefs), and acts in the environment to further its goals or objectives. In principle, any physical or abstract entity having intelligence, perception and actuation capabilities can be modelled as an agent. The evolution of software engineering along object oriented methodologies from design to implementation was instrumental in realizing agent models as computer software in a transparent manner.

The agent paradigm, though for apparently different reasons, came about in the midst of the macro-to-micro transition in simulation approaches. Agent based modelling is well fitting to individual based modelling and hence by extension to micro simulation. In many fields of science, the entities of interest in a simulation can be modelled as intelligent agents capable of acting autonomously. The modelling is more natural than previous approaches as it takes a one-to-one mapping between the real world entities and software agents. As a result of this, researchers in social and natural sciences are increasingly adopting agent based modelling and simulation as an appropriate paradigm in their respective domains. Software implementations of MABS in economics, sociology, biology, ecology, etc, demonstrate that MABS is a powerful tool for understanding and predicting the behaviour of dynamic systems. With MABS getting a wider audience in the simulation community, some argue that agent based modelling and simulation is a third way of doing science (inductive and deductive reasoning being the other two) [10].

In this thesis, we investigate MABS in distributed execution environments. MABS is a type of agent based simulation where several agents, each of which representing an entity in the simulated system, participate in a simulation. MABS is a micro-simulation methodology used to study evolutionary behaviour and emergent phenomena in complex systems constituted by autonomous entities. A defining characteristic of these entities is that they exhibit social behaviour, i.e., interact with each other according to rules that dictate their role and purpose in the system. It is argued that MABS is replacing or complementing macro-simulation techniques that use generalized models represented in the form of differential equations to describe system dynamics [20].

In the papers of this thesis, the terms agent based modelling, agent based social simulation (ABSS), ABS and MABS are interchangeably used to refer to the context of simulations involving several agents that interact (communicate) with each other to simulate emergent and evolutionary phenomena. MABS is a discrete event simulation (DES) technique that can be used to give insight into the dynamics of social phenomena and project

scenarios by formulating and testing models of existing phenomena through computational reconstruction [16]. According to Sansores et al.:

"Agent based modelling and simulation facilitates the study of how social phenomena emerge, that is, how the interactions and varied behaviours of individual agents produce structures and patterns. Thus agent based modelling is well suited for studying systems that are composed of interacting agents and exhibit properties arising from the interactions of the agents that cannot be deduced simply by aggregating the properties of individual agents." [17]

The dynamics of the system desired to be studied with MABS is captured at the micro level by modelling the individual beings that constitute the entities of the system as computational agents. As discussed earlier, these agents have sensory and perceptual capabilities, can pursue their objectives and make decisions autonomously, and interact with other agents. In some cases, the interaction can be according to predefined social and organizational structures. In other situations, however, the pattern of interaction evolves as the simulation proceeds and is an emergent phenomenon which is in fact, of primary interest to the researcher.

An important feature of agents that makes them appealing for the simulation of complex systems is that the behaviour of the simulated entity can easily be mapped into an agent's computational body. Other than the collective emergent phenomena, the simulation can show the behaviour at the individual agent level as well. As Kotal et al. put it,

"The simulation model preserves the granularity of the simulation at the individual level and at the same time is scalable and can simulate combined behaviour of huge crowds. [19]

1.2.1. Multi-Agent Based Simulation Tools

There are many ways for building agent based simulations. However, constructing MABS applications from scratch is a time-consuming and error-prone task. Instead, MABS researchers employ tools in the form of dedicated agent-based simulation software and general-purpose multi-agent platforms that handle the common and generic tasks. These tools aid researchers in the process of modelling and development of simulation applications [14]. The tools provide templates for modelling generic agent behaviour, manage

launching of the simulation, delivery of messages, scheduling and timing of events, provide mechanisms for life-cycle management, etc.

A high level overview of the mechanics of running a simulation on a MABS tool involves:

1. Setting up and initialization of the tool in a hosting environment;
2. Creating instances of simulation agents by modifying the supplied templates according to the roles they are intended to model;
3. Implementing additional features of the agents and the communication behaviour by writing program codes as necessary;
4. Enforcing time management mechanisms if there isn't one supported by the tool.

In some social systems, the entire phenomena, i.e., all of the entities, their behaviour and interactions need to be captured in the simulation model. This is because for such systems, the findings of partial simulations cannot always be generalized and do not produce meaningful results. A good example for this is the spread of epidemic in a given community or vehicular traffic in an urban center. It is difficult to draw conclusions by extrapolating the outcomes of a simulation that captures only a part of the whole. On this issue, Kota et al. further state that:

"To achieve realistic simulation, we should ideally be able to model the activities of every individual, even when there are hundreds of thousands of such individuals." [19].

In such situations, a large-scale MABS containing thousands or millions of agents is the result of the subsequent application development activity. A desktop computer cannot usually furnish such applications with the necessary computing power, leaving distributed processing as the only viable option. Large-scale MABS can thus be realized as parallel distributed simulations. Gasser et al. state that:

"Simulation of information intensive large-scale agent systems necessitates high computational power. Often, it is the only feasible form of computation to distribute the simulation model, and to run agents in parallel on multiple computational nodes, to gain the required performance speed up." [13]

One of the initial tasks of this research was to select a convenient tool or tools to be used for performance studies in a distributed environment. Tobias and Hoffman [11] evaluate freely available tools based on Java. Their work considers several factors ranging from usability to implementation issues. A survey of agent based modelling tools by Nikolai and Madey [12] provides a classification of over 50 products. The tools are categorized by programming language of origin, visual modelling features, support for parallelization, run-time environment, licensing requirements, documentation and support, etc. A survey was also conducted as part of this research to identify existing deployments and potential application areas of distributed agent based simulations.

From the perspective of this thesis, the choice of a simulation tool should consider the following factors:

1. Programming language: It is intended to execute the application in a distributed environment composed of heterogeneous resources. For this purpose, it is desired that the application be developed in a platform independent language. It was therefore decided that Java based tools will be considered for this work.
2. Parallelizability: Some simulation tools are simple desktop applications with no possibility to be parallelized. It is not necessary to consider such tools however wide their deployment base is. Accordingly, only tools with inherent parallelism or potential thereof are considered.
3. Documentation: availability of clearly written documentation and tutorials, in particular clear description of the APIs is essential to justify the potential usage of the tool.

Accordingly, one dedicated MABS tool called NetLogo and one general-purpose distributed multi-agent platform are chosen for this research.

1.2.2. Execution Models of Distributed MABS Applications

To understand the implementation aspect of distributed (or parallelized) MABS applications, it is convenient to first study how such applications are deployed on multi-agent platforms. The following discussion shows the execution model of a distributed MABS in this context.

At the computational level, the program codes of agents representing the simulated entities are realized as executable threads. The number of thread

instances thus corresponds to the number of entities simulated. The interaction between the real world entities takes the form of inter-agent communication. The messaging may be intra- or inter-node depending on where the communicating agents are deployed. As shall be seen later, the inter-node communication overhead of distributed MABS is a major source of performance bottleneck that needs to be addressed.

A simple model of a MABS execution cycle starts with the creation of agents in the platform. Agents expedite their tasks in two important phases: computation and communication phase. In the computation phase, agents execute program codes that correspond to the role they imitate. During the communication phase, agents send to and receive the latest messages from their peers as required by the system dynamics.

A critical aspect of any parallel or distributed simulation is time management, i.e., preserving the temporal characteristics of the simulated phenomena. The simulation programmer should ensure that the ordering and causality of events is maintained. Failure to achieve this would make the repeatability of the simulation experiment, and hence its validity, questionable. Modelling causality in general is a challenging task when simultaneous events are to be executed on a single processor. Although multi-threading creates an illusion of concurrency, the actual pieces of code are executed sequentially. Unless timing of events is handled with care, it will be impossible to honour the causality commitment. As will be shown later in this work, besides its implementation complexity, ensuring causality contributes a significant overhead both on application execution time and resource requirements.

The time management aspect of discrete simulations is distinguished by the mechanism used in the application to advance simulation time:

- Event-driven simulation: each event has a time-stamp associated with it that indicates the point in simulation time when the event occurs, and the simulation time is advanced to the time of the next event.
- Time-driven simulation: the simulation time is divided into a sequence of equal-sized time steps, and the simulation advances from one time step to the next;

Event-driven simulation is the more efficient of the two approaches both resource wise and time wise. However, a central coordinator is needed to keep track of the sequence of events and their executions, a requirement that

runs counter to the raison d'être of MABS [18]. This thesis mainly considers time-driven simulation as it is the most natural and direct way of mapping real world systems to MABS models.

1.3. Motivation and Research Questions

Distributed computing is developing fast with the availability and affordability of powerful computers. Newly emerged and existing applications benefited immensely from the rich pool of computer resources that can be tapped through this technology. Unfortunately, not all applications are blessed to exploit the full potential of distributed computing. One of such applications is agent based simulation.

It has been a while since the importance of agent based simulation in scientific research and management is recognized. Several studies applied MABS to understand and explain social dynamics and to predict emergent phenomena. However, most of the MABS applications were limited to desktop environment only. Although there are compelling reasons to port MABS to distributed environment, it could not be received with enthusiasm in the distributed computing community due to its poor performance characteristics. As a result, an entire group of problems that could have been studied with MABS, are either completely left in the wilderness or, only downsized versions of the actual problems have to be dealt with.

The principal objective of this thesis is to help simulation researchers to build large-scale simulation models that can run efficiently in distributed environment. By understanding the execution characteristics of MABS applications and identifying their performance bottlenecks, this thesis proposes some approaches to overcome these problems. The results of this research are mainly useful to simulation application architects, developers and other distributed systems professionals.

A number of tasks have been carried out in this research to answer this question. All publications, except Publication I describe these tasks and the results achieved. Publication I contains important findings that were important to build the necessary foundation for this research and gain an appreciation of the field of agent based simulation. In order to propose solutions to the identified problem, it is essential to understand the causes of the problem clearly. In this case, it means understanding the performance

characteristics of MABS in distributed environment and to quantifying their effects through performance models.

The next natural step would then be to work on possible solution alternatives to overcome the problems and make an evaluation of the alternatives to choose the most appropriate ones for different types of application scenarios. Accordingly, the following set of research questions were addressed in this thesis work:

RQ1: *What are the general characteristics of MABS applications?*

RQ2: *What are the major performance problems when executing a MABS application in a distributed environment?*

RQ3: *How can the performance of distributed MABS applications be predicted?*

RQ4: *How can MABS applications be efficiently executed in a distributed environment?*

Attempts have been made to address these questions in this research and interesting results were obtained. The contributions of this research and a brief summary of the results are presented in this chapter. These contributions are explained after giving a brief discussion on the methodology applied in this work.

1.4. Methodology

The main findings of this research are obtained through experimental work. However, during the initial phase, survey methods were applied to gain basic understanding of the problem. Through literature survey, the necessary background on the application areas of agent based simulation, the state-of-the-art in distributed computing, the various MABS tools and distributed technologies were studied.

To address RQ1, publications describing actual implementations or implementation tools for existing MABS applications were collected and

analyzed. A comprehensive evaluation framework was prepared to categorize MABS applications for this purpose.

Experimental work was performed to tackle RQ2. A Grid testbed was built and used to study the behaviour of MABS applications. A synthetic MABS application capturing the essential features of MABS was built using the web services technology and deployed on the testbed.

The same testbed was used to address RQ3 at the first stage. At later times, the execution environment was extended to include remote clusters and desktop machines. The experimental MABS application was modified to measure cause-effect relationships and to quantify the influences of the features identified in RQ2. Measurement data were collected according to the factorial design method [15] since we considered multiple independent variables affecting application execution characteristics. Empirical performance models were developed and validated for different infrastructure configurations and application scenarios using the collected measurement data.

Several performance improvement approaches were studied and solutions were proposed to deal with the problems in RQ4 based on the observations made during answering RQ2. Additional performance bottlenecks were discovered in different use cases and deployment models of MABS. Performance improvement approaches based on prediction models and heuristics are proposed.

1.5. Contributions of the Research

The publications included in this thesis deal with the four research questions. The contribution of each publication is presented in this section.

Publication I

This chapter presents a survey of MABS applications and addresses RQ1. The findings of the survey deal with a wide range of issues beyond this research question. They show the importance of MABS in the study of social, organizational, economic, biological, etc., systems. The findings also

show that in some domains, the simulation requires the deployment of a large number of agents. However, the publishers of the surveyed papers carried out only partial simulations to study the problems even though such studies demanded full-scale simulations. It can be argued that this is possibly due to lack of computing resources, appropriate simulation tools and methodologies. The latter case was more evident in many publications where the simulation applications are built from scratch. It also follows that instances of large-scale MABS deployments are yet to be seen despite the need for having one in certain application domains. This finding served as an incentive to conduct this research as explained in the Motivation section.

Publication II

The work presented in this chapter addresses RQ2, and to some extent, RQ4. It is based on experiments performed on a Grid testbed using a synthetic MABS application as a workload. It has the following three important contributions:

1. It investigates the performance characteristics of MABS applications in distributed environment. It is found that agent task granularity, cross-node communication among agents and time synchronization are the major factors influencing application performance. This result served as a stepping stone in finding performance optimization strategies.
2. It proposes a middleware architecture for improving the performance of MABS applications. The proposed middleware is specifically adapted for MABS that execute on FIPA compliant multi-agent platforms [22]. It conforms to the generic Grid Monitoring Architecture, but contains additional blocks for application tuning and load balancing as well.
3. It demonstrates how the middleware can be used to improve performance by reducing communication overhead through re-allocation of agents according to their interaction pattern and peer groupings.

Publication III

This work mainly addresses RQ3. Its main contribution is building performance models for distributed MABS applications. As in Publication II, an application workload featuring the fundamental characteristics of MABS

was designed and deployed on a Grid testbed. The model is used to give an estimate of application execution time for a given workload configuration. The configuration specifies the nature of the workload (simulation size task granularity and communication behaviour). The prediction model can be used for scheduling and load balancing purposes.

This work provides partial answers to RQ4 as well. With the help of the developed prediction model, it would be possible to estimate the number of computers needed to optimally deploy a MABS application with a given configuration. Additionally, the prediction model was used to validate the work in Publication II, in which performance is improved by reducing communication overhead through agent re-allocation.

Publication IV

The focus of this work is RQ4, application performance. It investigates scalability issues in distributed multi-agent platforms used for hosting MABS applications. The Java Agent Development Framework (JADE) deployed in a cluster environment was used for this experiment. It investigates RQ2 by extending the study in publication II to validate the findings on sources of performance bottlenecks. On top of those identified in Publication II, multi-agent platforms have an additional performance problem. The directory services of the platforms are centralized and were not designed in anticipation of large-scale agent deployment. The significance of this problem became evident when it was observed that the execution time rises exponentially as the simulation size approaches a thousand agents, regardless of the number of computers used. This work proposes replication and caching of directory information to solve this problem and evaluates different caching approaches. A remarkable improvement in performance was recorded with the proposed approach.

Another important contribution of this work is reducing communication overhead for large-scale MABS deployed on JADE. Like its directory service, the message transport of JADE is not optimized to cope with the communication needs of thousands of agents interacting simultaneously. A solution is proposed and tested in this work to handle the communication among agents located on the same computer, bypassing the built-in messaging service of JADE. The improvement achieved through the proposed optimization is observed to be significant.

Publication V

This chapter addresses RQ3 and RQ4. The main difference between this and previous works is that, performance studies are performed in different execution environments. Additionally, the performance characteristics of different distributed MABS architectures were investigated. These architectures represent the two prominent interaction patterns among agents: peer-to-peer and hierarchical.

Answers to RQ3 are presented through the several performance models developed for the two interaction architectures in different execution environments. The models were also used to predict the efficiency of the optimization approaches proposed in Publication IV under different execution scenarios.

One contribution to answer RQ4 in this work appears in performance optimization through message aggregation. Performance models were built to predict and compare execution times using JADE's built-in transport services and the proposed message aggregation strategy. The predictor is important to make application tuning decisions, whether or not message aggregation is appropriate for a given MABS configuration.

Publication VI

Question RQ4 is the target of this work. It studies performance improvement through better management of simulation time. In Publication II, it was discovered that time step synchronization contributes significant overhead in a distributed simulation. A novel algorithm for optimistic synchronization in MABS applications is proposed and evaluated.

A global virtual time is maintained across all computers participating in the simulation. The individual agents, however, may have different local times, based on the number of simulation time steps they have executed. As opposed to conservative synchronization, agents are allowed to advance multiple time steps until the time allocated by the Operating System (quantum) to the agent's thread expires. Messages exchanged between agents bear a time-stamp whose value is equal to the local time of the sending agent.

In many event-driven distributed simulations, rollback is allowed if out of time messages are delivered. However, in large-scale MABS, because the

communication graph is dense, the cost of rollback far outweighs the benefit. The cost of rollback was investigated and experimentally validated in this work. The algorithm, however, allows rollback in limited cases depending on the dynamics of the simulated system and its interaction graph.

Although agents are allowed to advance their local time, there is a limit in the number of time steps an agent can be ahead of the global virtual time. Messages sent from advancing agents to lagging ones are saved as future messages by the lagging agents on receipt. These messages are stored until the recipient agent's local time advances to the timestamp they bear. This would demand additional memory for storing future messages. The algorithm gives an estimate of the memory requirement as a function of the simulation size. The most important contribution of this work is that, it is possible to improve execution time at the cost of memory capacity. MABS implementers can make relevant decisions, whether or not to adopt the proposed optimistic synchronization subject to availability of memory.

Publication VII

This publication addresses RQ3 and RQ4. It presents an extensive investigation of the following distinct approaches to implement distributed MABS.

1. Parallelization of existing desktop ABSS tools.
2. Optimization of parallel multi-agent platforms for MABS
3. Adoption of the web services technology for MABS

In the first case, the NetLogo simulation package was used. Although this tool is not a parallel application, it was possible to exploit some of its features to create multiple instances of the simulation on different machines. The partitioning of the simulation space, the exchange of data between adjacent partitions (if needed) and synchronization issues were studied. The proposed strategy in this work can be used to parallelize large-scale optimization simulations as parameter sweep applications. The application model is master-slave, a client machine assigns tasks to several worker machines. The same set of simulation configuration is executed on the workers. The optimization search space is partitioned and assigned to each worker according to its capacity. The workers would then return partial results and report how far they have exhausted the search space assigned to them so far.

Load balancing measures are taken by the master to repartition the remaining search space based on performance feedback obtained. The model can be recursively applied for cluster and Grid execution environments.

Another use case studied under the above category is that of large-scale spatially explicit simulations. A similar logic as the parameter-sweep case can be applied to partition the simulation space. However, there is an additional requirement here that communications are needed between adjacent nodes to update the states across their overlapping regions. This would impose a restriction on synchronizing the simulation time at all workers. Algorithms have been proposed to minimize the synchronization overhead and to improve performance by applying the concept of pipelining as in implicit parallelism on single processor computers. The pipelining approach reduces the idle time that would have to be wasted while adjacent nodes update each other.

The work also contributes partial answers to RQ1. It identifies simulation use cases that can be efficiently executed on parallel multi-agent platforms. These platforms run individual agents in their own threads. For large-scale simulations, this would lead to massive level of multi-threading whose overhead is unjustifiable for MABS characterized by fine-grain computational tasks. In stead, it is more reasonable to simulate coarse-grained MABS where the issue of autonomy, complex communications, intelligence and learning cannot be handled by other means. MABS models for markets, supply chains, transportation, etc., can be good candidates for this simulation environment. The performance issues that were addressed in the previous publications were revisited in this work.

The last part of this work proposes a new execution model for agent based simulations based on the web services concept. Although web services have been around for quite a while, they were not considered as MABS implementation alternatives because of performance problems. However, with the emergence of the service oriented computing paradigm, it would be appropriate to think of the future of MABS along this direction. Additionally, the speed of interconnection networks is improving. Another incentive is, when a distributed application is deployed in the Cloud, the computer resources on which it runs are often virtual nodes located on the same physical machine. The network latency in such set up is very small and MABS applications can run more efficiently here.

Publication VIII

This work addresses RQ4. One of the findings in Publication II was to reduce communication overhead by reallocating agents according to their interaction patterns. By reallocation, in this context, it is meant to migrate agents across nodes. There are some difficulties with this issue. The first one is that the interaction pattern is only known after agents are deployed on different nodes and the simulation progressed over some time. The second challenge is that migration is a computationally expensive operation. The main problem is, however, that obtaining the interaction pattern is a graph theory problem. Working with communication graphs may need computer time comparable with the actual simulation task itself.

In this work, an algorithm is proposed to study the interaction pattern and generate a cost effective reallocation strategy. The algorithm employs heuristics to generate an optimized migration scheme and evaluates a performance prediction model to know if the cost of reallocation is offsetted by the anticipated reduction in communication overhead. If not, it will generate another scheme with less migration cost, and re-evaluate the prediction model. This process is iterated through until a workable scheme is achieved and the migration is then effected.

The most significant contribution of this work is that the heuristics amortizes the cost of processing the agent communication graph over a longer time. There is idle time at all nodes due to synchronization requirements as explained earlier. The algorithm utilizes this idle time to update communication graphs iteratively. The migration scheme is therefore produced at virtually no visible computational cost although extra memory is required to store and process the communication graph.

1.6. Validity

Encouraging results were obtained in this work, particularly in the areas of performance analysis and parallelization of single threaded MABS applications. However, there are also limitations that need to be addressed or acknowledged. These limitations or threats to validity of the results are customarily grouped into two, external and internal, [21] are discussed below.

1.6.1 External Threats (Generalizability)

It may not be possible to generalize the results of this work to all MABS domains due to the following considerations.

Limitation of the application workload used to build performance models: Synthetic workloads with generic features representing MABS applications were used in most experiments. Several simplifying assumptions and idealizations were allowed in the workload model. The issue of I/O overhead, which is too large to be ignored in most simulations, was not accounted for in the models.

Limitation of execution environment: Many of the experiments were not conducted in a production environment where the communication and computation characteristics may significantly differ from the experimental environment. The measurements were mostly taken on Grid and cluster test beds specifically set up for this purpose. The models must be validated over a larger number of nodes than were available in the testbeds.

1.6.2 Internal Threats

The internal validity threats are divided into three areas, associated with the instrumentation process, accuracy of measurement data and the analysis of measured data.

Quality of measurement data:

It was difficult to suppress interferences of other factors during the experiment. The effects of Operating System overheads such as cache memory and I/O management, etc, are not accounted for. Attempts were, however, made in the design of the experiments to keep the effect of these factors at the minimum possible.

Time measurements were not precise enough for fine-grained simulations. These types of MABS have a small computational task, the duration of which is difficult too capture accurately using languages such as Java. The instrumentation overhead appears to exceed the agents' computation time. One of the difficulties with fine-grained simulations is in setting the granularity parameter as desired. This has a serious impact on the accuracy of the performance prediction model.

Because MABS applications are not deterministic, the computational granularity of agent tasks cannot often be constant. The amount of variation in the granularity affects the measurements to a certain extent. The effect of this variation is difficult to account for in concrete terms.

Quantity of measurement data:

For the prediction models to be comprehensive enough, performance measurements have to be taken for a wide range of input parameters. The measured data used in this work is sufficient to make predictions over a limited range only. The input space is a combination of 4 parameters that can take a large span of values in real world simulations.

To ensure the reliability of the data, it was necessary to take repetitive measurements. The threat of measurement consistency is considered to be less significant in comparison with the other factors cited above.

Method of Analysis:

The mathematical methods used in the prediction scheme employ linearization techniques which are not always precise enough. The performance models were built using the generalized least squares (GLS) method which assumes system linearity. Combined with the inaccuracy in time measurements and the limitation of input data, it is difficult to generalize the performance models for all MABS use cases and the results should be mainly used as guidelines.

1.7. Conclusions

This thesis addresses the research questions identified in this research by making two important contributions for efficient execution of distributed MABS. One aspect of the contributions is the identification of the performance characteristics of MABS and the subsequent development of prediction models for application optimization. The other contribution is, the thesis proposes algorithms and heuristics to reduce communication and synchronization overheads which were earlier identified as the dominant

performance bottlenecks. The research questions were tackled through surveys and experimental works as presented in the resulting publications.

The question of identifying large-scale MABSs as formulated in RQ1 was addressed in Publication I. The summary of the surveyed applications shows that the problem (simulation) size in certain phenomena necessitates the use of distributed MABS as the only feasible alternative to study the phenomena.

The performance characteristics of MABS applications as expressed in RQ2 were studied in Publication II. Although the factors considered as performance bottlenecks in this work are common to most parallel and distributed applications, the thesis gives useful insights on the relative influence of communication overhead and task granularity in the particular application of interest. The findings were essential to tackle the remaining research questions.

To address the problem established in RQ3, performance modelling and prediction of MABS applications were performed in different types of distributed environment. The tasks carried out to deal with this problem and the results obtained in Publications III, V and VII show that the performance models give reasonable estimates of application run time and task allocation for load balancing purposes.

The ultimate objective of this thesis is improving application performance by addressing RQ4. All publications except the first deal with this question, albeit in different ways. Performance model based optimizations are presented n Publications III, V and VII. Algorithmic approaches for application performance improvement are given in Publications VI and VIII. A simple heuristics to improve performance by reducing communication overhead is given in Publication II. Some approaches to tune distributed multi-agent platforms by addressing problems inherent to their architectural designs and implementations is presented in Publication IV.

All optimizations proposed in this work are intended to bring about performance improvement without affecting the configuration of the computing and communication resources on which the application runs. Such application level optimizations are useful for application developers and users who have no rights or privileges to do resource level optimizations, i.e., not authorized to alter the configurations of remote cluster or Grid resources.

Performance modelling of MABS application has some challenges. First, for those finely-grained MABS applications, it is difficult to build accurate

performance prediction models. Second, MABS are not like traditional applications with predictable behaviour. The amount of work performed in one unit of time by two real world entities does not translate to equivalent amount of computational work. It is therefore important to understand the nature of the MABS very well before applying the prediction models on it.

The choice of a simulation tool is an important step in a MABS application design. A design that leads to a fine-grained, massively multi-threaded application with a dense communication graph is the worst workload in a distributed environment. It follows from this that not all MABS applications are good candidates for distribution.

The Grid is suitable for loosely coupled application such as workflows. As such, it is not a preferable environment for fine-grained MABS. However, if the only available alternative is the Grid, then optimization techniques such as message aggregation and agent reallocation have to be employed. The Cloud environment seems to be well suited for MABS. Because the computer resources availed by Cloud service providers often exist on the same physical machine or are connected by high speed network backbones, the communication overhead is usually low. However, Cloud Computing is envisioned to work as a service platform and MABS applications need to be adapted to this paradigm.

1.8. Future Work

The thesis covered important aspects of distributed MABS applications. However, there are several areas of improvement that need to be considered to achieve the objective of making MABS a feasible candidate application for distributed systems.

The performance prediction models built in this research are too general and need to be refined further for each category of MABS applications. For example, market models differ in application behaviour from crowd simulation. The interaction pattern, the frequency and types of messages, the granularities of the tasks, etc, are quite different. If the performance models are customized taking this variations into account they can be very useful. The workloads used in our performance models capture the kernel of the MABS. Visualization components that are features of many MABS tools are

not included in this model. The application models should therefore be expanded to incorporate visualization features and the associated performance factors.

One area where more research should be done is the service enablement of MABS applications. There is a wide acceptance that the computing ground is shifting in favour of service oriented applications. The execution infrastructure is also provisioned as a service. Since many simulation researchers are unlikely to own high performance resources, they will have to depend on computing power leased from service providers for their large-scale simulations. In view of all these, it is imperative to redefine the architecture of MABS applications to adapt to the ongoing computing service delivery models. Accordingly, the proposed MABS platform realization with web services needs to be further studied.

Another technological shift in the computing field is the appearance of GPUs with many cores at reasonable prices. These devices outperformed conventional CPUs in certain simulations applications [24]. Although the memory bandwidth problems of GPUs makes them less than ideal for some applications, fine-grained MABS applications can benefit greatly from these devices. Many similar applications in chemistry and biology were reportedly able to achieve several times speedup when ported to GPU. Research towards porting existing simulation tools to GPU should be one area to be considered.

1.9. References

[1] Foster, I. and Kesselman, C. (eds) "*The Grid: Blueprint for a Future Computing Infrastructure*" Morgan Kaufmann Publishers, San Fransisco, USA. 2003.

[2] Jarvis, S.A. et al. "*Performance prediction and its use in parallel and distributed computing systems*" Proceedings of the 17th International Symposium on Parallel and Distributed Processing. IEEE Computer Society. Washington DC, USA. 2003. pp. 276.1

[3] Tanenbaum A.S., Steen M. *"Distributed Systems Principles and Paradigms"* Second Edition, Pearson Prentice Hall, USA 2007. ISBN 0-13-613553-6

[4] Max K.G. *"Network Distributed Computing: Fitscapes and Fallacies"*, Prentice Hall Publishers, 2004. ISBN- 13:978-0131001527

[5] Stockinger, H. *"Defining the grid: a snapshot on the current view"* The Journal of Supercomputing, Springer Netherlands, vol. 42, Issue 1, Pages 3-17, 2007.

[6] Zanikolas, S. and Sakellariou R. *"A taxonomy of grid monitoring systems"* Elsevier Journal of Future Generation Computer Systems vol. 21, 2005. pp. 163–188

[7] Massie, M.L. et al. *"The ganglia distributed monitoring system: design, implementation, and experience"* Elsevier Journal of Parallel Computing Vol. 30, 2004. pp. 817–840

[8] Nabrzyski, J., et al. "*Grid Resource Management: State of the art and Future Trends.*" Kluwer Academic Publishers, Norwell, MA, USA. 2004.

[9] Birman, K.P. *"Reliable Distributed SystemsTechnologies, Web Services, and Applications"*, Springer Verlag, 2005. ISBN- 13:978-0-387-21509-9

[10]Macal, C.M. and North, M.J. *"Tutorial on Agent based modeling and simulation*" Proceedings of the 2005 Winter Simulation Conference. M.E. Kuhl, N.M.Steiger, et. al. (eds.), 2005.

[11] Tobias, R. and Hofmann, C. *"Evaluation of free Java-libraries for social-scientific agent based simulation"* Journal of Artificial Societies and Social Simulation vol. 7, No. 1, 2004.

[12] Nikolai, C. and Madey G. *"Tools of the Trade: A Survey of Various Agent Based Modeling Platforms"* Journal of Artificial Societies and Social Simulation Vol. 12, No. 2(2), 2009.

[13] Gasser, L., Kakugwa, K., Chee, B, and Esteva M. *"Smooth scaling ahead: Progressive MAS simulation from single PCs to Grids"* Lecture Notes in Computer Science, Springer Berlin/Heidelberg. vol. 3415, 2005. pp. 1-10

[14] Maria, A. "*Introduction to modeling and simulation*" Proceedings of 29th Conference of Winter Simulation. Atlanta, USA. 1997. pp 7-13

[15] Berling, T. And Runeson, P. *"Efficient Evaluation of Multifactor Dependent System Performance Using Fractional Factorial Design,"* IEEE Trans. on Software Engineering, vol. 29, no. 9, 2003. pp. 769-781

[16]Conte, R., Gilbert, N., and Sichman, S. "*MAS and social simulation: A suitable commitment*" Lecture Notes in Computer Science, Springer Berlin/Heidelberg. vol. 1532, 1998. pp. 1-9

[17] Sansores, C., Pavon, J. and Gomez-Sanz, J. "*Visual modeling for complex agent-based simulation systems*" Lecture Notes in Computer Science, Springer Berlin/Heidelberg. vol. 3891, 2006. pp. 174-189

[18] Davidsson, P. *"Multi Agent Based Simulation: Beyond Social Simulation"*, In Moss, S. and Davidsson, P. (Eds.) Multi Agent Based Simulation, Springer Verlag LNCS series, vol. 1979, pp. 141-155

[19]Kota, R., Bansal, V., and Karlapalem, K. "*System issues in crowd simulation using massively multi-agent systems*" Proceedings of International Student Workshop on Agents. Kyoto, Japan, 2006.

[20] Bonabeau, E. *"Agent-based modeling: Methods and techniques for simulating human systems"*, Proc. National Academy of Sciences of the USA. Vol. 99, No. supp 3 2002. pp. 7280-7287

[21] Robson C. "*Real world research*" Second Edition. Blackwell Publishers, USA, 2002.

[22] The Foundation of Intelligent Physical Agents (IEEE SC) www.fipa.org.

[23] Weiss, G. Multi Agent Systems: "*A modern approach to distributed artificial intelligence.*" MIT Press, England, 2000.

[24] Marandi, V. and Dinavahi, V. *"Large-scale transient stability simulation on graphic processing units"* Proc. The IEEE Power & Energy Society, 2009.

[25] Wang, F., Turner, S.J., and Wang, L. "*Agent communication in distributed simulations*" Lecture Notes in Computer Science, Springer Berlin/Heidelberg. vol. 3415, 2005. pp. 11-24

TWO

Applications of Multi Agent Based Simulation

Paul Davidsson, Johan Holmgren, Hans Kyhlbäck, Dawit Mengistu, Marie Persson

School of Engineering, Blekinge Institute of Technology
Soft Center, 372 25 Ronneby, Sweden

Abstract

This paper provides a survey and analysis of applications of Multi Agent Based Simulation (MABS). A framework for describing and assessing the applications is presented and systematically applied. A general conclusion from the study is that even if MABS seems a promising approach to many problems involving simulation of complex systems of interacting entities, it seems as the full potential of the agent concept and previous research and development within MABS often is not utilized. We illustrate this by providing some concrete examples. Another conclusion is that important information of the application, in particular concerning the implementation of the simulator, was missing in many papers. As an attempt to improve this situation, we provide some guidelines for writing MABS application papers.

1 Introduction

The research area of Multi Agent Based Simulation (MABS) continues to produce techniques, tools, and methods. In addition, a large number of applications of MABS have been developed. By MABS application we here mean actual computer simulations based on agent-based modelling of a real (or imagined) system in order to solve a concrete problem. The aim of this paper is to present a consistent view of MABS applications and to identify trends, similarities and differences, as well as issues that may need further investigation.

As several hundreds of MABS applications have been reported in different publications, we had to make a sample of these. After having performed a preliminary search for papers describing MABS applications that resulted in about 50 papers, we identified one publication that was dominating. About 30% of the papers were published in the post-proceedings of the MABS workshop series [1, 2, 3, 4, 5] whereas the next most frequent publications covered only 10% of the papers. We then chose to focus on this publication series and found 28 papers containing MABS applications (out of 73). Even if we cannot guarantee that this is an unbiased sample, we think that selecting all the applications reported in a particular publication series with a general MABS focus (rather than specializing in particular domains etc.), is at least an attempt to achieve this.

In the next section, we present the framework that will be used to classify and assess the applications. This is followed by a systematic survey of the sampled papers. Finally, we analyze our findings and present some conclusions.

2 Evaluation framework

A MABS application models and simulates some real system composed of a set of entities. The MABS itself can be seen as a multi-agent system composed of a set of (software) agents. That is, there is a correspondance between the real system and the multi-agent system as well as between the (real) entities and the (software) agents. We will use the terms “system” and

"entity" when referring to reality and "multi-agent system" and "agent" when referring to models and simulations.

For each paper we describe the problem studied, the modeling approach taken to solve it, the implementation of the simulator, and how the results are assessed.

2.1 Problem description

Each problem description includes the domain studied and its purpose.

2.1.1 Domain

The domain of an application refers to the type of system being simulated. We identified the following domains after analyzing the sampled papers:

1) An *animal society* is composed of a number of interacting animals, such as an ant colony or a colony of birds. The purpose of a simulation could be to better understand the individual behaviors that cause emergent phenomena, e.g., the behavior of flocks of birds.
2) A *physiological system* is composed of functional organs integrated and co-operatively related in living organisms, e.g., subsystems of the human body . The purpose could be to verify theories, e.g., the regulation of the glucose-insulin metabolism inside the human body..
3) A *social system* consists of a set of human individuals with individual goals, i.e., the goal of different individuals may be conflicting. An example could be to study how social structures like segregation evolve.
4) An organization is here defined as a structure of persons related to each other in purposefully accomplishing work or some other kind of activity, i.e., the persons of the organization have common goals. The aim of a simulation could be to evaluate different approaches to scheduling work tasks with the purpose of speeding up the completion of business processes.
5) An economic system is an organized structure in which actors (individuals, groups, or enterprises) are trading goods or services on a market. The applications which we consider under this domain may be used to analyze the interactions and activities of entities in the system to help understand how the market or economy evolves over time and how

the participants of the system react to the changing economic policies of the environment where the system is operating.

6) In an ecological system animals and/or plants are living and developing together in a relationship to each others and in dependence of the environment. The purpose could be to estimate the effects of a plant disease incursion in an agricultural region.

7) A physical system is a collection of passive entities following only physical laws. For example, a pile of sand and the purpose of the simulation may be to calculate the static equilibrium of a pile considering forces between beads and properties within the pile considered as a unit.

8) A robotic system consists of one or more electro-mechanical entities having sensory, decision, tactile and rotary capabilities. An example is the use of a set of robots in patrolling tasks. The purpose of the simulation could be to study the effectiveness of a given patrolling strategy.

9) Transportation & traffic systems concern the movement of people, goods or information in a transportation infrastructure such as a road network or a telecommunication network. A typical example is a set of interacting drivers in a road network. The purpose of a simulation could be to create realistic models of human drivers to be used in a driving simulator.

2.1.2 Purpose

The purpose of the studied MABS applications is classified according to prediction, verification, training and analysis. We refer to prediction as making prognoses concerning future states. Verification concerns the purposes of determining whether a theory, model, hypothesis, or software is correct. Analysis refers to the purpose of gaining deeper knowledge and understanding of a certain domain, i.e., there is no specific theory, model etc to be verified but we want to study different phenomena, which may however lead to theory refinement. Finally, training is for the purpose of improving a person's skills in a certain domain.

The following examples in the area of oil consumption are given to point out distinctions between these categories. We classify; making prognoses of oil consumption in a country as prediction, how the oil consumption varies according to different tax levels as analysis, letting decision makers handle simulated oil shortage scenarios as training, and testing whether existing

hypotheses concerning decreased oil consumption when the price of alternative fuels decreases as verification.

2.1.3 End-users

The end-users of a MABS application is the intended users of the simulator, e.g., scientists, policy makers, managers, or professionals.

2.2 Modeling Approach

The modeling aspects are captured by the seven aspects described below.

2.2.1 Simulated Entities

They are entities distinguished as the key constituents of the studied systems and modeled as agents. Four different categories of entities are identified: Living thing - humans or animals, Physical entity - artifacts, like a machine or a robot, or natural objects, Software process - executing program code, or Organization - an enterprise, a group of persons, and other entities composed by a set of individuals.

2.2.2 Number of Agent Types

Depending on the nature of the studied application, the investigators have used one or more different agent types to model the distinct entities of the domain.

2.2.3 Communication

The entities can have some or no interaction with one another. The interactions take place in the form of inter-agent communication, i.e., messaging. Here, we defined two values to indicate whether communication between agents exists or not.

2.2.4 Spatial Explicitness and Mobility

Spatial explicitness refers to the assumption of a location in the physical space for the simulated entities. This can be expressed either as absolute

distance or relative positions between entities. Mobility refers to the ability of an entity to change position in the physical space. Although the real world entities may be spatially situated or moving from place to place, this fact need not be considered in the simulation if its inclusion or omission does not affect the outcome of the study.

2.2.5 Adaptivity

Adaptivity is the ability of the entities to learn and improve with experience that they may acquire through their lifetime. Two values are defined to indicate whether the simulated entities are adaptive or not.

2.2.6 Structure of MAS

The structure of MAS refers to the arrangement of agents and their interaction in the modeled system to carry out their objectives. This arrangement could be in one of the following three forms: peer-to-peer, hierarchical, or recursive (figure 1). In a peer-to-peer arrangement, individual entities of the modeled system are potentially interacting with all other entities. In a hierarchical structure, agents are arranged in a tree-like structure where there is a central entity that interacts with a number of other entities which are located one level down in the hierarchy. Each low level entity in turn interacts with a number of other lower-level entities and so on. Whereas, in a recursive structure, entities are arranged in groups, where the organization of each group could be in either of the forms mentioned above, and these groups of entities are interacting among each other to accomplish their tasks.

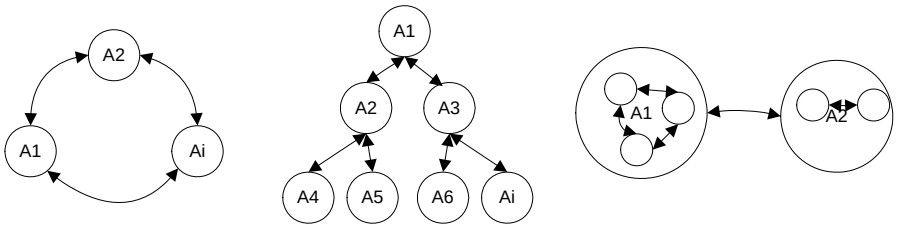


Figure 1. Structure of MAS: peer-to-peer, hierarchical, and recursive.

2.2.7 Dynamic

If the modeled entities are able to come into existence at different instances of time during a simulation, we regard them as dynamic.

2.3 Implementation Approach

The implementation approach used is described in terms of four aspects: the platform used for the implementation, the (number) size of agents in the implementation, the agent execution environment, and their mobility.

2.3.1 Platform used

The software platform is the development environment, tool or language with which MABS is developed. The platforms provide support to different degrees for the developers so that they need not worry about every implementation detail.

2.3.2 Simulation size

The size describes the number of agents participating in the implementation of the MABS application. If the number is different between simulations or is changing dynamically during a simulation, we will use the largest number.

2.3.3 Distributed

MABS applications, depending on the size and sometimes the nature of the application, may require different execution environments: a single computer, if the number is small or on several computers in a distributed environment, if the number of agents is large.

2.3.4 Mobile agents

Agents executing in a distributed environment can be described by their mobility, as static or mobile. Static agents run on a singular computer during their lifetime. Mobile agents, on the other hand, are able to migrate, between computers in a network environment.

2.4 Results

The classification of the result of the approaches will be in terms of maturity of the research, comparison to other approaches and the validation performed.

2.4.1 Maturity

MABS applications can have varying degree of maturity. In our classification framework the lowest degree of maturity is conceptual proposal. Here the idea or the principles of a proposed application is described, but there is no implemented simulator. The next level in the classification is laboratory experiments. Here the application has been tested in a laboratory environment. The data used in the experiment can either be real data, i.e. taken from existing systems in the real world, or data that is not real, i.e. artificial, synthetic or generated. Further, the type of data has been divided into limited/partial or full-scale data. The full-scale data represents data for a whole system, while the limited/partial data only covers parts of the system. The final level, deployed system, indicates that the MABS system actually is or has been used by the intended real end-users, e.g., traffic managers that use a simulator for deciding how to redirect the traffic when an accident has ocurred.

2.4.2 Evaluation comparison

If a new approach is developed to solve a problem which has been solved previously using other approaches, the new approach should be compared to existing approaches. That is, answer the question whether MABS actually is an appropriate approach to solve the problem. Such an evaluation could be either qualitative, by comparing the characteristics of the approaches, or quantitative, by different types of experiments.

2.4.3 Validation

In order to confirm that a MABS correctly models the real system it needs to be validated. This can be performed in different ways, qualitatively, e.g., by letting domain experts examine the simulation model, or quantitatively, e.g., by comparing the output produced by the simulator with actual measurements on the real system.

2.4 Summary of Framework

The table below summarizes the framework. Note that for some applications it is necessary to use several categories to properly describe an aspect of that application.

	Aspect	**Categories**
Problem description	Domain	1. Animal societies 2. Physiological systems 3. Social systems 4. Organizations 5. Economic systems 6. Ecological systems 7. Physical systems 8. Robotic systems 9. Transportation/traffic systems
	End-user	1. Scientists 2. Policy makers 3. Managers 4. Professionals
	Purpose	1. Prediction 2. Verification 3. Analysis 4. Training
Modeling approach	Simulated entity	1. living thing 2. physical artefact 3. software process 4. organisation
	Number of agent types	1 - 1.000
	Communication	1. no 2. yes
	Spatial explicitness	1. no 2. yes
	Mobility	1. no 2. yes
	Adaptivity	1. no 2. yes
	Structure (of MAS)	1. peer-to-peer 2. hierachical 3. recursive
	Dynamic	1. no 2. yes
Implementation approach	Platform used	NetLogo, RePast, Swarm, JADE, C++, etc.

	Aspect	Categories
	Simulation size	1 - 10.000.000
	Distributed	1. no 2. yes
	Mobile agents	1. no 2. yes
Results	Maturity	1. Conceptual proposal 2. Laboratory experiment 2.1. artificial data 2.1.1. limited/partial 2.1.2. full-scale 2.2. real data 2.2.1. limited/partial 2.2.2. full-scale 3. Deployed system
	Evaluation comparison	1. None 2. Qualitative 3. Quantitative
	Validation	1. None 2. Qualitative 3. Quantitative

3 Results

Below is a table where the papers are classified according to the framework. If it does not explicitly state to which category the simulator belongs but there are good reasons to believe that it belongs to a particular category, it is marked by an asterisk (*). If we have not managed to make an educated guess, it is marked by "-".

Paper	Problem			Modeling								Implementation				Results		
	Domain	End-user	Purpose	Sim. Entity	N.o. types	Commun.	Spatial	Mobile	Adaptive	MAS str.	Dynamic	Platform	Size	Distributed	Mobile	Maturity	Evaluation	Validation
[6]	4		1	3	2	2	1	1	1	2*	1	C++	10	1*	1*	2.1.1	1	1
[7]	4		3	1	-	2	2	-	2	-	-	-	-	-	-	1	1	1
[8]	4		1,3	1	4	1	1	1	1*	1*	1	-	-	1*	1*	2.2.1	1	3*
[9]	4		3	1,4	2	1	1	1	1	2*	1	RePast	60	1*	1*	2.1.1	1	1
[10]	9		1	2	-	1	2	2	1	-	1	-	120	-	-	2.1.2	1	1
[11]	3		3	1	1	1	2	1	1	1	1	-	100	1*	1*	2.1.1	1	2*
[12]	3,9		2	1,2	3	2	2	2	1	1	2	-	12000	2*	2*	2.2.2	1	1
[13]	4		3	1	2	2	2	2	2	1	2	WEA	25*	2*	2*	2.2.2	1	3
[14]	9		2	1	1	1	2	2	1*	1	1*	-	100*	1*	1*	2.1.1	2	3
[15]	3,6		3	1	3	1*	2	2	2*	1	2	Swarm	540	1*	1*	2.1.1	1	1
[16]	5,9		3	1	6	2	1	1	1	2	1	Jade	7	1*	1*	2.2.1	1	1
[17]	7		3	2	1	2	2	1	1	1	1*	-	10^6	1*	1*	2.1.1	2,3	2
[18]	5		2,3	1,4	3	2	1	1	1*	2	2	-	102	1*	1*	2.1.1	1	2
[19]	3		2	1	1	1	2	2	1*	1	2	NetLogo	200	1*	1*	2.1.1	2	1
[20]	1		3	1	2	1*	2	2	1	1	1	ObjectPascal	8	1	1	2.1.1	1	3
[21]	3		2	1	1	1*	2	2	1	1	1	-	250	1*	1*	2.1.1	1	1
[22]	2		2	2	3	2	1	1	2	3	1	Java	4	2*	1*	2.2.1	1	3
[23]	3		3	1	3	2	1	1	1	1	1	-	9	1*	1*	2.1.1	1	1
[24]	3		3	1	1	2	2	2	1	1	2	Sugarscape	700	1*	1*	2.1.1	1	1
[25]	3,6		3	1	3	2	2	2	1	1	1	Cormas	-	1*	1*	2.2.1	1	3
[26]	3		3	1,3	3	2*	1	1	1	1	2	VisualBasic	10000	1	1	2.1.1	3	1
[27]	4,7		3	1,2	5	2	2	2	2	1	1	C++	1	1	1	2.1.1	1	1
[28]	3		2,3	1	2	1	1	1	2	1	1	NetLogo	500	1*	1*	2.1.1	2	1
[29]	4		3	1,2	3	2	2	2	2	1	1	RePast	61	1*	1*	2.2.1	2	1
[30]	8		1,2	2	1	2	2	2	1	1	1	C++	25	1*	1*	2.1.1	1	1
[31]	5		3	1	7	2	1	1	2	2	1	DECAF	3	1*	1*	2.1.1	1	1
[32]	3		3	1	1*	2	2	1	2*	1	1	-	-	1*	1	2.1.1	2	2
[33]	5		3	1	1	2	1	1	1	1	1	-	24*	1*	1*	2.1.2	3	1

4 Analysis

4.1 Problem description

The results indicate that MABS is often used to study systems involving interacting human decision makers, e.g., in social, organizational, economic, traffic and transport systems. This is not surprising given the fact that qualities like autonomy, communication, planning, etc., often are presented as characteristic of software agents (as well as of human beings). However, as (some of) these qualities are present also in other living entities, it is interesting to note that there was only one paper on simulating animal societies and just two involving ecological systems.

Very few papers are found on simulating technical systems, such as ICT systems, i.e., integrated systems of computers, communication technology, software, data, and the people who manage and use them, critical infrastructures, power systems etc.. The aim of such models might be to study and have a deeper understanding of the existing and emerging functionalities of the system and analyze the impact of parameter changes. (The only paper on simulating technical systems concerned robotic systems.)

The most common purpose of the applications included in the study was analysis. However, no paper reported the use of MABS for training purposes indicating that this may be an underdeveloped area.

4.2 Modeling Approach

The simulated entities are mostly living things, indicating that MABS is believed to be better suited to model the complexity of human (and animal) behaviour compared to other techniques. However, it should be noted that in some applications there were entities not modeled and simulated and implemented as agents. Hybrid systems of this kind are motivated by the fact that some entities are passive and are not making any decisions, especially in socio-technical systems.

The model design choices for some of the aspects seem to be consequences of the characteristics of the systems simulated. After all, the aim is to mirror the real system. These aspects include number of agent types, only about 15% of the applications had more than three different agent types, spatial explicitness (60% do use it), mobility of entities (50%),

communication between entities (64%), and the structure of the MAS where a vast majority used a peer-to-peer structure (77%).

However, there are some modelling aspects where the strengths of the agent approach do not seem to have been explored to its full potential. For instance, only 9 of the 28 papers make use of adaptivity, and just 7 out of the 27 implemented systems seem to use dynamic simulations.

4.3 Implementation Approach

Nearly half of the papers do not state which software were used to develop the MABS. In particular, it is interesting to note that the two papers with the largest number of agents do not state this. Of the agent platforms and simulation tools available, none is dominantly used. In fact, many of the simulations were implemented with C++ or programs developed from scratch. A possible reason for this may be that many MABS tools and platforms make limiting assumptions regarding the way that entities are modeled and therefore difficult to use.

The number of agents in the simulation experiments is typically quite small (see figure 2). In 50% of the papers the number of agents were 61 or less. The fact that most simulation experiments were limited covering only a part of the simulated system, may be an explanation for this. Moreover, there may be a "trade-off" between the complexity of the agents and the number of agents in the experiments, i.e., that large sized simulations use relatively simple agents whereas smaller simulations use more complex agents. However, further analysis is necessary before any conclusions can be drawn.

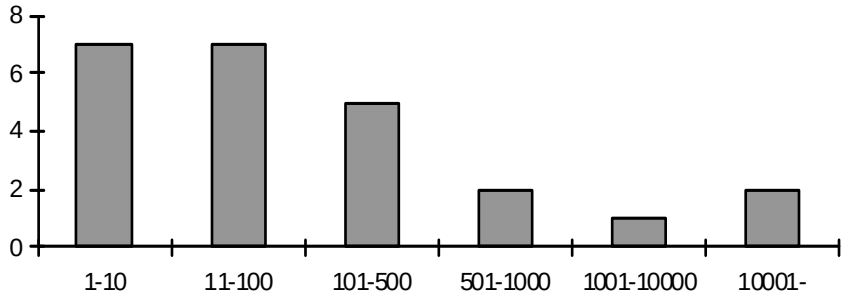


Figure 2. The frequency of different simulation sizes (number of agents).

It seems as very few of the simulators are distributed, and no one is using mobile agents. However, these issues are seldom described in the papers.

4.4 Results

We have not encountered any MABS applications that are reported to be deployed to solve actual real world operational tasks. The examples of depolyed systems are limited to the cases where the researchers themselves are the end-users. The cause of this could be the fact that MABS is a quite new methodology, or that the deployment phase is not described in scientific publications. Moreover, very few full-size simulation experiments are documented. The reasons for this are seldom discussed in the papers but are probably lack of computing hardware, software (such as proper agent simulation platforms), or the time available to perform the experiments.

Many of the simulation experiments are conducted with artificial data, typically making simplifying assumptions. This is often due to reasons beyond the researchers' control, such as availability of data. As a consequence, it may be difficult to assess the relevance of the findings of such simulations to the real world problems they aim to solve.

Less than half of the simulations are actually reported to be validated. This is particularly striking as it is in most cases the complex behaviors of humans that are being simulated.

4.5 Limitations of the Study

Although the conclusions drawn from our study are valid for the work published in the MABS proceedings, a larger sample is probably needed to verify that they hold for the whole MABS area. For instance, we have encountered papers in other publications that cover domains that are not represented in our sample.

There were a number of interesting aspects that we were not able to include in our study. For example, regarding the problem description, the size of the actual problem, i.e., the system being simulated would be interesting to know. Typically, only a partial simulation is made, i.e., the number of entities in the real system is much larger than the number of agents in the simulation.

However, in most papers the size of the real system is not described and often it was very difficult for us to estimate the size.

Another interesting aspect not included in this study is the modeling of entities. The representation of the behavior and state of the real world entities should be sufficiently sophisticated to capture the aspects relevant for the problem studied. We initially categorized the ways of modeling the entities in the following categories: Mathematical models; Cellular automata; Rule-based (a set of explicit rules describe the behavior of the agent); Deliberative (the behavior is determined by some kind of reasoning such as planning). Unfortunately, there were often not enough information in the papers concerning this aspect. Related to this is the distinction between proactive versus reactive modeling of entities, which also was very difficult to extract from the papers due to lack of information. Regarding the implementation, we wanted to investigate how the agent models were implemented in the simulation software. We found examples ranging from simple feature vectors (as used in traditional dynamic micro simulation) to sophisticated software entities corresponding to separate threads or processes. However, also in this case important information was often left out from the presentation.

5 Conclusions

The applications reviewed in this study suggest that MABS seems a promising approach to many problems involving simulating complex systems of interacting entities. However, it seems as the full potential of the agent concept often is not utilized, for instance, with respect to adaptivity and dynamicity. Also, existing MABS tools and platforms are seldom used and instead the simulation software is developed from scratch using a ordinary programming language. There may be many reasons for this, e.g., that they are difficult to use and adopt to the problem studied, or that the awareness of the existence of these tools and platforms is limited.

Something that made this study difficult was that important information, especially concerning the implementation of the simulator, was missing in many papers. This makes it harder to reproduce the experiments and to build upon the results in further advancing the state-of-the-art of MABS. A positive effect of our study would be if researchers became more explicit and clear about how they have dealt with the different aspects that we have used in the

analysis. Therefore, we suggest the following check-list for MABS application papers:

1. Clearly describe the purpose of the application and the intended end-users.
2. Indicate the typical size of the system (that is simulated) in terms of entities corresponding to agents.
3. For each agent type in the simulation model, describe
   a. what kind of entities it is simulating,
   b. how they are modelled (mathematical, rule-based, deliberative, etc.),
   c. whether they are proactive or not,
   d. whether they are communicating with other agents or not,
   e. whether they are given a spatial position, and if so, whether they are mobile, and
   f. whether they are capable of learning or not.
4. Describe the structure of the collection of agents, and state whether this collection is static or agents can be added/removed during a simulation.
5. State which simulation (or agent) platform was used, or in the case the simulator was implemented from scratch, what programming language was used.
6. State the size of the simulation in terms of number of agents.
7. Describe how the agents were implemented; feature vectors, mobile agents, or something in-between.
8. State whether the simulator actually has been used the intended end-users, or just in laboratory experiments. In the latter case indicate whether artificial or real data was used.
9. Describe how the simulator has been validated.
10. Describe if and how the suggested approach has been compared to other approaches.

Future work includes extending the study using a larger sample, e.g., include other relevant workshops and conferences, such as Agent-Based Simulation, and journals such as JASSS, and making a comparative study with more traditional simulation techniques including aspects such as size, validation, etc.

6 References

1. S. Moss, P. Davidsson (Eds.), *Multi-Agent Based Simulation*, LNAI Vol. 1979, Springer, 2000.
2. J. S. Sichman, F. Bousquet, and P. Davidsson (Eds.), *Multi-Agent Based Simulation II*, LNAI Vol. 2581, Springer, 2003.
3. D. Hales et al. (Eds.), *Multi-Agent Based Simulation III*, LNAI Vol. 2927, Springer, 2003.
4. P. Davidsson, B. Logan, K. Takadama (Eds.), *Multi-Agent and Multi-Agent-Based Simulation*, LNAI Vol. 3415, Springer, 2005.
5. L. Antunes, J.S. Sichman (Eds.), *Multi-Agent Based Simulation 2005*, To be published in the LNAI series, Springer, 2006.
6. E. Kafeza, K. Karlapalem, Speeding Up CapBasED-AMS Activities through Multi-Agent Scheduling, in [1]
7. T. Wickenberg, P. Davidsson, On Multi Agent Based Simulation of Software Development Processes, in [2]
8. J. Rouchier, S. Thoyer, Modelling a European Decision Making Process with Heterogeneous Public Opinion and Lobbying: The Case of the Authorization Procedure for Placing Genetically Modified Organisms on the Market, in [3]
9. D. A. Robertson, The Strategy Hypercube: Exploring Strategy Space Using Agent-Based Models, in [3]
10. I. Noda, M. Ohta, K. Shinoda, Y. Kumada, H. Nakashima, Evaluation of Usability of Dial-a-Ride Systems by Social Simulation, in [3]
11. R. Sosa, J. S. Gero, Social change: exploring design influence, in [3]
12. K. Miyashita, SAP: Agent-based Simulator for Amusement Park -Toward eluding social congestions through ubiquitous scheduling, in [4]
13. A. P. Shah, A. R. Pritchett, Work Environment Analysis: Environment Centric Multi-Agent Simulation for Design of Socio-technical Systems, in [4]
14. S. El hadouaj, A. Drogoul, S. Espié, How to Combine Reactive and Anticipation: The Case of Conflicts Resolution in a Simulated Road Traffic, in [1]
15. L.S. Premo, Patchiness and Prosociality: An Agent-Based model of Plio/Pleistocene Hominid Food Sharing, in [4]

16. M. Bergkvist, P. Davidsson, J. A. Persson, L. Ramstedt, A Hybrid Micro-Simulator for Determining the Effects of Governmental Control Policies on Transport Chains, in [4]
17. L. Breton, J.-D. Zucker, E. Clément, A Multi-Agent Based Simulation of Sand Piles in a Static Equilibrium, in [1]
18. I. Takahashi, I. Okada, Monetary Policy and Banks' Loan Supply Rules to Harness Asset Bubbles and Crashes, in [3]
19. C. M. Henein, T. White, Agent Based Modelling of Forces in Crowds, in [4]
20. C. K. Hemelrijk, Sexual Attraction and Inter-sexual Dominance among Virtual Agents, in [1]
21. R. Pedone, R. Conte, The Simmel Effect: Imitation and Avoidance in Social Hierarchies, in [1]
22. F. Amigoni, N. Gatti, On the Simulation for Physiological Processes, in [2]
23. M. R. Rodrigues, A. C. da Rocha Costa, Using Qualitative Exchange Values to Improve the Modelling of Social Interaction, in [3]
24. S. Tomita, A. Namatame, Bilateral Tradings with and without Strategic Thinking, in [3]
25. L. Elliston, R. Hinde, A. Yainshet, Plant Disease Incursion Management, in [4]
26. P. Winoto, A Simulation of the Market for Offenses in Mulitagent Systems: Is Zero Crime Rates Attainable?, in [2]
27. N. Sahli, B. Moulin, Agent-based geo-simulation to support human planning and spatial cognition, in [5]
28. L. Antunes, J. Balsa, P. Urbano, L. Moniz, C. R. Palma, Tax compliance in a simulated heterogeneous multi-agent society, in [5]
29. A. Melo, M. Belchior and V. Furtado, Analyzing police patrol routes with the simulation of the physical reorganization of agents, in [5]
30. A. Machado, G. Ramalho, J.-D. Zucker, A. Drogoul, Multi-Agent Patrolling: an Empirical Analysis of Alternative Architectures, in [2]
31. F. McGeary, K. Decker, Modeling a Virtual Food Court Using DECAF, in [1]
32. T. Downing, S. Moss, C. Pahl-Worstl, Integrated Assessment: Prospects for Understanding Climate Policy Using Participatory Agent-Based Social Simulation, in [1]
33. E. Ebenhöh, Modeling Non-linear Common-pool Resource Experiments with Boundedly Rational Agents, in [5]

THREE

Middleware Support for Performance Improvement of MABS Applications in the Grid Environment

Dawit Mengistu, Paul Davidsson, Lars Lundberg
Department of Systems and Software Engineering, School of Engineering,
Blekinge Institute of Technology,
372 25 Ronneby, Sweden
E-mail: {dawit.mengistu, paul.davidsson, lars.lundberg}@bth.se

Abstract

The computational Grid is an infrastructure which enables the execution of applications demanding huge computing resources. Hence, it can be the right environment for large-scale Multi-agent based simulation (MABS) applications. However, due to the nature of the Grid and the characteristics of MABS, achieving optimum performance poses a great challenge. Performance study of MABS applications is therefore a necessary undertaking which requires an understanding of these characteristics and the extent of their influence. Moreover, owing to the dynamicity and heterogeneity of the Grid, it is difficult to achieve performance gains without a middleware support for application deployment and dynamic reconfiguration. This paper presents a study of the key features of MABS applications that affect performance and proposes a supportive middleware to MABS platforms. Experiments show that the proposed middleware can bring performance improvement for MABS applications on the Grid.

Keywords: *MABS, Performance Improvement, Grid Computing, Middleware.*

1 Introduction

The computational Grid is a hardware and software infrastructure that provides access to a possibly huge pool of heterogeneous computing resources. It is a resource rich distributed computing platform, that can support the execution of complex applications in various fields. Applications in climatic simulations, earth science, medical research, etc., which would take a lot of time or be unmanageable in other environments can now be executed on the Grid within a reasonable time. One major area where the benefit of Grid computing is evident is in the field of simulation, particularly in multi-agent based simulation (MABS).

MABS applications involve modeling real world entities as software agents that interact with each other. In most of the cases, MABS applications are limited, partial, conducted on a smaller scale, with fewer agents, making use of many simplifying assumptions, etc [2]. MABS models often are designed as down-sized versions of the actual problem, or are implemented as separately run components executed at different time and/or premises, due to lack of computing and visualization resources.

Recent developments in computing have created an opportunity to make advances in large scale MABS. High performance computing resources such as the Grid have become available for solving problems that were once thought to be intractable. Agent programming tools and multi-agent platforms are developed to deploy multi-agent applications on distributed resources with less programming effort. Although multi-agent platforms designed for distributed computing are maturing, the work done so far to use them in the Grid environment is very limited.

One challenge when implementing MABS applications on the Grid concerns the application performance that is caused by the relatively high communication-to-computation ratio of multi-agent systems. There are also other inherent features that affect the performance of MABS when it is deployed as a distributed application in heterogeneous environment such as the Grid.

The purpose of this work is to study the performance characteristics of MABS applications on the computational Grid and propose a middleware to be used for performance improvement. Using a synthetic MABS application as a workload, we conducted an experiment to study the behaviour of MABS, investigate application design issues, and ways of overcoming performance

bottlenecks. We also developed a prototype of the middleware which was used in the experiment. The rest of this paper is organized as follows: the motivations to study performance issues for MABS deployment on the Grid is presented next, followed by a discussion on performance issues of MABS applications and the Grid. We then present an architectural framework of the proposed middleware and an overview of the envisaged MABS deployment in the Grid. In Section 5, the experimental environment and the characteristics of the workload used are explained. Section 6 presents the results of the experiment with a qualitative discussion and the performance improvement achieved using a prototype of the middleware. We conclude the paper by summarizing the findings of our study and citing directions for future work.

2. Motivations for This Study

Although there are many circumstances where partial simulations suffice to understand the real world phenomena, there exist a number of problems that cannot be adequately understood with limited-scale simulations [1]. Partial simulations may not always yield complete results, require additional time and effort to assemble, analyze and interpret. The domain being studied can be better understood if a large, or even full scale simulation where each real world entity is modeled as an agent and the relations between the entities is represented accordingly. In problems where the number and type of entities is large, it will be very difficult to represent the dynamics in its entirety due to lack of adequate computing resources. A very large scale simulation would require the availability of computational power large enough to run the simulation within an acceptable span of time.

Technically, a MABS application may be implemented as a multi agent system comprising several agents working in parallel performing computing, communication and data access tasks asynchronously. Therefore, MABS models can be conveniently ported to the Grid as parallel applications. In particular, the Grid can be an ideal computational environment for MABS because:

1. Simulations involving thousands or millions of agents and highly complex and data intensive tasks can benefit from the Grid's high throughput resources.
2. The distributed nature of agent-based computing makes distributed platforms like the Grid a natural working environment for multi-agent systems.

However, MABS applications designed without properly considering their execution characteristics (particularly the communication behaviour) in a distributed computing environment are unlikely to be good candidates for the Grid. It is therefore necessary to identify the key features of MABS applications that affect performance on the Grid. By understanding the impact of these features, isolating and eliminating the performance bottlenecks they cause, simulations can be executed efficiently. A MABS platform supported by a middleware that monitors the execution environment and adapt the application for best performance on the Grid is desired. A performance-aware middleware that works with the MABS platform to support dynamic application tuning and optimization for the Grid is required to achieve this.

3. Performance of MABS Applications on the Grid

A MABS platform provides the physical structure in which agents are deployed. It generally consists of agent support software, agent management system, a directory service, a message transport service and the agents themselves [15]. The architecture of the application running on the platform affects performance significantly.

3.1. Key Features of MABS Affecting Performance

Although it is not possible to list all items that potentially contribute to performance problems, we have identified relevant key features of MABS to be examined in this experiment. The following architectural features have significant influence on the performance of MABS applications.

1. Multi-threading. The autonomous behaviour of agents is conveniently realized by writing the simulation application in such a way that each

agent runs within its own thread of control. Platforms that do not provide multi-threading may hamper modeling MABS applications in distributed environments. If the simulation size is large, i.e., involves too many agents, it would require substantial multi-threading. This undermines system performance severely since running too many threads causes a lot of context switching, more overhead and complicates scheduling tasks.

2. High communication-to-computation ratio. Agents simulating real world entities should mimic the interactions among the entities they model. The interaction is inter-agent messaging, where the message originating and destination agents are deployed on the same or different nodes. Messaging between agents running on the same node is essentially data movement within the same physical memory, while that between agents on different nodes involves local or wide area communication. A key requirement of the simulated problem may be that the agent originating a message not perform any action (hence should be in a waiting state without executing any useful operation) until it receives the reply to the message it sent out. If intense interaction between agents running on separate nodes is involved, this forced waiting undermines performance.

3. Time-step synchronization. For an agent based simulation, to reflect the chronological sequence of events in the simulated problem and guarantee causality of actions, the timing of the agent executions throughout the simulation is almost all cases of great importance. This is normally carried out by forcing a currently running thread to yield involuntarily, to avoid running out of synchronism with the other agents. The yielding thread will stay in the wait state until all agents are brought to the same temporal situation. If the size of the task executed within a unit of time is very small (i.e., the application is fine grained), this would entail a lot of context switching overhead which may even exceed the actual useful computation. This problem may further be exacerbated depending on the scheduling policy of the run-time environment.

The above factors may occur in combination, making performance analysis of MABS applications very complex. Another inherent feature of agent based applications affecting performance is that unlike object oriented applications, execution time cannot be predicted from the size of the executable code alone. Deploying such applications on a dynamic execution environment like the Grid will make the problem even more complex. The MABS application would therefore require proper architectural considerations.

3.2. Performance Analysis of Grid Applications

Performance analysis for efficient application execution and scheduling is a major challenge to Grid application developers [13]. Although recent Grid monitoring tools and schedulers come with improved features, they do not adapt to application specific architectural details in many cases due to their generic features. The performance of domain specific Grid applications can be enhanced by using application specific middleware which assist Grid schedulers. A performance-aware-middleware supports application modeling, prediction, optimization based on information acquired from Grid monitoring services. However, due to the dynamic behaviour of the Grid and the heterogeneity of resources, the implementation of performance analysis techniques in the context of Grid computing is more complex than in other environments. Because of this, static performance support tools do not provide effective support. As a result, the use of a middleware for dynamic performance analysis and prediction is becoming more common. Our plan to use a middleware support for MABS applications is therefore based on the inherent nature of the Grid environment

4. Proposed Architecture

The proposed Grid middleware is to be used as an interface between a MABS platform and the Grid resources and services. As explained earlier, the major purpose of employing this layer is to optimize a MABS application run-time code to execute efficiently in a given Grid environment. This middleware can interact with Grid monitoring services to improve performance using the information it receives from the services. In a Grid environment where monitoring tools are not available or accessible, the middleware can perform several useful tasks such as:

- Partitioning the simulation application into balanced tasks that can be dispatched to available Grid nodes. The simulation size is determined by the number of entities simulated or the number of agents to be deployed. Balancing tasks corresponds to dividing the number of agents equally into the number of Grid nodes taking part in the simulation.
- Distributing the task to the nodes. The task to be run on a node is deployed as a web service launched through the middleware.

- Monitoring the performance of the application. The middleware collects application information such as messaging statistics, thread waiting times, etc., from the Grid nodes through web service invocations.
- Reallocates agents and migrating tasks to improve performance. Based on the application performance data collected from Grid nodes, the middleware will decide on an optimal job redistribution strategy and reallocate the tasks to the nodes as necessary.

When implemented in conjunction with Grid infrastructure monitoring tools, the Middleware will interface the platform functionalities with the Grid resources. It uses performance data collected by Grid monitoring services for performance modeling and prediction purposes. It will then carry out essential tasks such as load balancing and reallocation strategies for job optimization and match making. The Grid scheduler will then be able to receive an optimized job matching predicted performance and execution requirements such as deadlines, if any. Figure 1 shows the architecture of the proposed performance aware middleware.

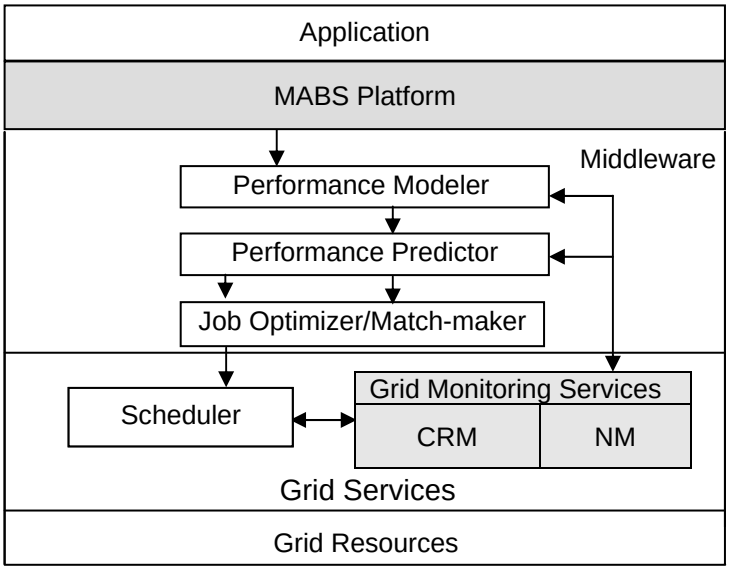


Fig. 1 The proposed middleware architecture.

The Simulation Modeler assists users by taking away most of the system level programming tasks from them so that they can focus only on the

functional logic of the simulated entities. According to [2], the simulation modeler is implemented as a customizable framework for modeling social phenomena in different domains. It should thus consist of a domain specific model editor and a user interface that enables capturing of the entities and the problem to be simulated itself.

The middleware consists of the following components:

1. The *Performance Modeler* uses performance data collected through Grid monitoring services, i.e., Compute Resource Monitoring (CRM) and Network Monitoring (NM), to build the execution performance model of the simulation workload. The model contains information needed to make scheduling decision, reallocation strategy, etc.
2. The *Performance Predictor* predicts the execution time of the simulation based on the output of the performance modeller and data from Grid monitoring services on current status of the Grid infrastructure. The prediction is bound to change with changes in the Grid environment. The predictor can also decide the optimal application deployment/ reallocation strategy.
3. The *Job Optimizer/ Match Maker* undertakes the necessary modifications on run-time parameters of the tasks for optimal use of the Grid resources on which each task is to be deployed. This component will thus assist the generic Grid scheduler to make more refined scheduling decisions.

The MABS platform consists of the components necessary to run a meaningful simulation application in a distributed environment. Figure 2 shows the components in detail.

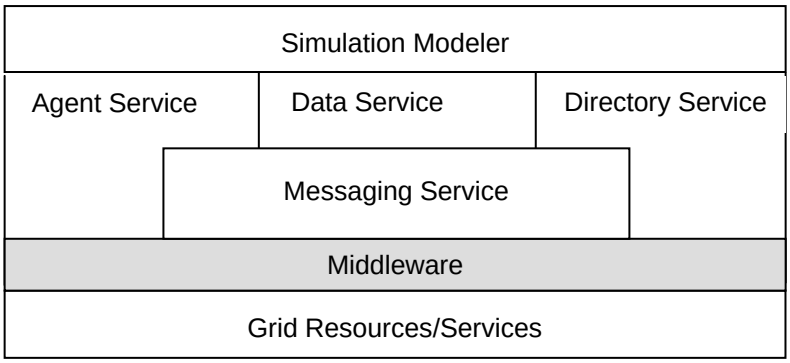


Fig. 2 Major components of a MABS application platform in the Grid environment

Since the Grid employs service oriented technologies, it is convenient to implement these platform components as distributed services to facilitate interoperability with other services.

The Directory Service manages information about the agents such as naming, locating, etc. Upon launching, agents automatically register in the directory service with a unique identifier. This service has a hierarchical architecture distributed over the Grid simulation nodes, with each node managing the information of the agents running on it. A central directory service is responsible for coordinating the directories of individual nodes.

The Data Service is an agent that keeps the repository of the data produced in the simulation. The data is converted into a knowledgebase which the agents regularly use to update their beliefs about the environment.

The Agent Service handles functionalities such as the launching and termination of agents, execution of agent code, registration and deregistration of agents, etc.

The Messaging Service is responsible for routing and delivery of messages between agents. In general, messages could be destined to peers residing on the same node (inbound) or elsewhere on the Grid (outbound). This service interacts with the Directory Service to decide whether messages are inbound or outbound.

5. Experiment

An experiment is designed to study the effect of the key MABS features discussed in Section 3.1 and to implement a middleware support for performance improvement. The MABS application used in the experiment is a synthetic simulation workload which incorporates these key features. The implementation details of the workload and the experimental environment are discussed next.

5.1. Workload Characterization

The workload is a multi-threaded MABS application. Each agent in the simulation is characterized by the following features:

- Has its own thread of execution and is recognized as a distinct process or thread by the underlying operating system;
- Communicates with other agents through inter-thread communication APIs implemented at the application level.
- A program code that captures the behaviour of the real world entities it is intended to represent.

For the purpose of this experiment, the agent code will have two components, computational and communication, clearly separated to manipulate the control variables and to study the effect of the earlier discussed features of MABS. The computational part mainly represents the role (task) of the agent, while the communication part handles the inter-agent communication activities. The actions of an agent are therefore realized by a thread code that performs a computation, followed by a messaging and yields to enforce the time-step synchronization.

If the workload is implemented on a stand-alone machine, since all threads run on that same machine, the inter-thread communication is essentially inbound. On a Grid version, however, the threads are launched on separate machines and the communication can be outbound too. The Grid workload should be partitioned into tasks of equal size, to be launched on the nodes.

The communication characteristics are of primary interest and should be well defined in the workload model. We defined the intensity of outbound communication as a parameter. We also study the effect of time-step synchronization in two situations. First, we see the extent to which the presence of time-step synchronization affects performance by taking two simulation runs with and without synchronization. We will then investigate the effect of performance for different levels of application granularity. In this experiment, the application granularity is considered to be the time it takes to execute one computational loop at the end of which synchronization between agents is enforced. Fine-grained applications perform smaller computational task in each time step while coarse-grained applications have larger size of code to be executed between successive time steps.

It is desired that performance instrumentation causes as little perturbation as possible to the instrumented application. Therefore, data manipulation and transfer tasks were deferred to the end of the workload execution not to interfere with the normal course of the application. In the case of the Grid, it is believed that putting measurement data on top of every messaging data will increase the overhead and should be avoided or minimized. Therefore, the measured data are stored in elementary data structures until the end of the simulation and communicated to the master node only thereafter.

The workload is modeled in terms of known and predictable computational and communication tasks to be carried out by each thread. The effect of known external factors that would bias the outcome should be minimized. For this reason, the application does not contain such tasks as I/O operations other than the communication explicitly required for inter-agent messaging.

An important requirement of the measurement process in multi-threaded applications is capturing the desired performance metrics with minimal overhead. To achieve this, individual threads maintain their respective copies of performance data, but share a single instrumentation code [8].

The validity of the experimental setup was verified, and show that the workload and performance metrics chosen conform to the envisaged objective and are equally valid for both the stand-alone and Grid environments. The workload was run in both environments several times to see if the experiment is repeatable and the outcomes are predictable. It was observed that factors that had not been accounted for have very little effect on the experiment and the workload model is indeed valid for the study.

5.2. Grid Environment

The Grid version of the experiment was conducted on a Globus (GT4) Grid testbed with machines having PIII 1000MHZ processors with 512MB RAM running Linux. These machines are connected via a 100Mbps switch to the Internet. Since the main purpose of the study is the effect of MABS characteristics on performance, it was necessary to run the experiment under a controlled environment to isolate and observe the effect of each of MABS features explained earlier. For this reason, the Grid nodes do not have

additional loads. Furthermore, some components of the Globus toolkit such as GFTP and security features, although installed, are not used here since their presence or absence bears no relevance to this experiment.

Each node hosts an identical copy of the MABS application as a set of Web Services deployed in its Globus container. Another machine used as a master node serves as a launching pad where the simulation application is divided into sub tasks to be invoked as web services on each Grid node.

The stand-alone version of the application was run on a Linux machine having identical configuration with the worker nodes used in the Grid version of this experiment. Since the applications run in both environments are functionally the same, we only discuss the Grid version in the rest of this paper.

5.3. Implementation of Workload

The workload is a Java multi-threaded application written in accordance with the requirements of the workload model explained in 5.1 above. The Grid version of the application is a web service launched from a master (client) node where the workload is partitioned into pieces of balanced sizes distributed to worker nodes on the Grid. For example, if the real world problem consists of 100 similar entities modeled by 100 agents and the simulation is run on 2 Grid nodes, the master node launches the web services on each worker node such that the worker hosts 50 agents. Inter-agent communications can occur between those agents residing on the same or different nodes.

The agents are instantiated as individual threads in the Grid web service. As explained earlier, the workload has a fixed size and is executed as computational loops interleaved with messaging operations. Since granularity is a parameter, simulation runs are conducted with different granularity levels. The execution time of one loop is thus a measure of the application's granularity. Inter-thread messaging normally takes place following this. The product of the granularity and the number of loops is constant and represents the size of the workload.

If an agent sends out a message and does not receive a reply immediately or within a reasonable time, it should yield control and wait until the reply

comes instead of advancing the computational loop. Since all agents take control of the CPU turn by turn, the expected reply will be received sooner or later.

Since the threads need to maintain coherence with respect to the simulated time, they should be made to advance execution of loops at a synchronized pace. If a thread finishes a task ahead of the others, it should stay in a waiting state by yielding control of the CPU until other threads come to the same level.

A shared memory area is reserved for managing the inter-agent messaging. Each agent would be able to read messages destined to it and also send out messages to others. If an inbound message is sent, (to an agent on the same node) it will be stored in the message buffer of the destination agent. If, however, it is destined to an agent on another node, it is kept in a different area from which the messaging web service collects outbound messages and delivers them to the client. The web service on the destination will then collect the message from the client through its messaging service and places it in the destination agent's messaging buffer.

The ratio of outbound messages to the total number of dispatched messages is an important parameter for studying the effect of network latency. We have therefore defined it as outbound (OB) communication ratio. The ratio is given as the number of messages sent out by an agent to agents residing on nodes other that hosting the sender, out of every 100 messages initiated by that agent, expressed in percentage.

Time-step synchronization is enforced at two levels, local and global as follows: first, all agents running on the same node are internally synchronized. When the node level (local) synchronization is completed, the master node is notified through a response to a Web Service request initiated by the master node itself. After all nodes reported their status to the master (global) in this way, they will receive another Web Service invocation as a command to proceed with the next time step or computational loop, for a specified number of times.

To launch the application, the master code is initiated on the client machine and invokes the Web service on each node by passing the required parameters of the simulation. This code also monitors the overall execution of the application and facilitates the delivery of outbound messages in cross node communications.

Using different values of the experiment parameters, i.e., simulation size (number of agents), granularity and outbound communication ratio, several runs of the experiment were conducted. The parameters were varied independently and together, to understand their individual and combined effects. The observations made from the experiment, the analysis of the measurement data and the performance improvement obtained using the middleware are discussed in the following section.

6. Results

The observations made from the experiment are presented below in this section according to the order in which they appear in Section 3.1. The plots from the repeated runs of the experiment summarize these observations. A prototype of the middleware proposed in Section 4 is implemented to see how it improves performance by addressing one of the factors, namely, the outbound communication. The performance gain with the middleware is presented at the end of this Section.

6.1. Effect of Multi-threading on Performance

If the program codes of individual agents are realized as separate threads, the application will be massively multi-threaded when the size of the simulation is large. Because such level of multi-threading causes a lot of overhead on the computational resource, execution time increases in a quadratic fashion and performance reduces significantly. To illustrate this, several runs of the experiment were conducted with different number of agents deployed as individual threads, taking part in the simulation. The outcome of the experiment is shown in figure 3.

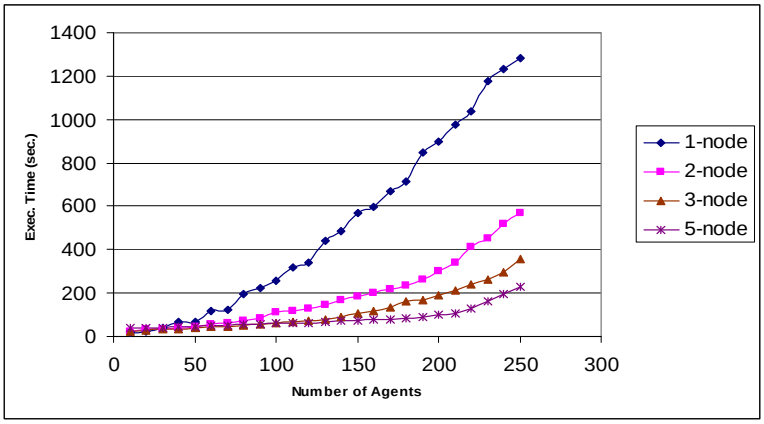


Fig.3. Execution time of MABS applications on the Grid increases non-linearly with simulation size.

As can be seen from the plot, implementing the agents as individual threads degrades performance significantly. It follows that, if the simulation is distributed over M nodes, given other conditions equal, the speed up gain in the best case can be as high as M2. Several reasons are attributed to this, such as poor performance of the JVM, operating system policies, presence of synchronized methods in the agent codes to ensure data integrity, etc.

6.2. Effect of outbound communication

The effect of outbound communication is typically discernible when the requirement of the simulation is that agents expecting replies for messages they sent out cannot proceed with their tasks until the reply arrives. In such cases, the agents will be forced to stay in a wait state until the reply arrives. The contribution of this waiting time could even be well larger than that of the time needed to execute the agent code depending on the intensity of the communication and the granularity of the MABS application. Coarse-grained applications have generally low communication-to-computation ratio and will not be affected as seriously as fine-grained ones. As the simulation size increases, however, other factors such as synchronization, multi-threading, etc., also come into the picture. The effect of outbound communication is felt most if the size of inter-agent messages is too large to fit into the communication buffers.

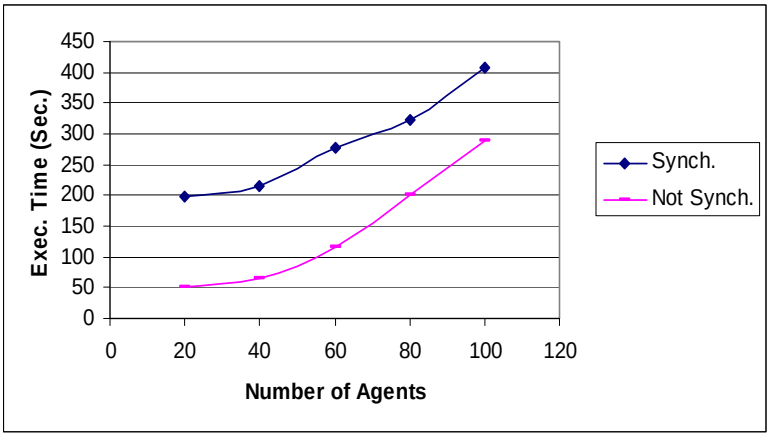
Instead of sending out messages instantly as they are produced, storing them in one location until the messaging web service collects them from that node for routing reduces network traffic significantly. This technique, known as message aggregation, is a very useful strategy employed in other communication intensive distributed applications also.

6.3. Effect of Time Step Synchronization

As explained earlier, the simulation cannot mimic the real world behaviour unless the operations of all simulating agents are synchronized in time steps. Time step synchronization undermines performance by introducing additional overhead as can be seen in the following figures.

The plots show that fine-grained simulations are affected most by time-step synchronization. Furthermore, the performance loss is more significant in small-scale simulations. For larger simulations however, the significance of this loss will be less since other factors related to the operating environment take a larger share of the performance drop. In limited cases, MABS applications may not need synchronization at all. Such cases arise when the simulated system does not involve any communication between entities or the chronological sequence of simulated events is of no interest. In such cases, it is possible to achieve significant performance gains by avoiding time-step synchronization.

a) Fine-grained MABS



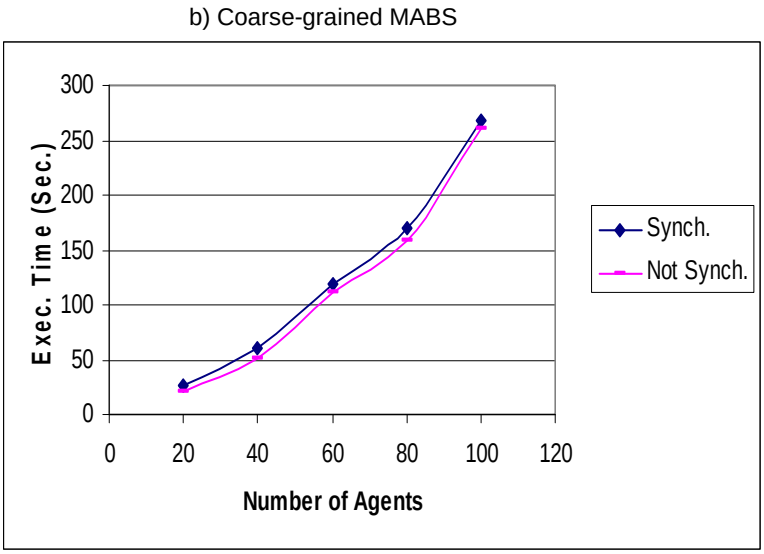


Fig. 4. Comparison of simulations with and without time-step synchronization on a 5-node Grid, with no outbound communications. Fine grained simulations are affected most.

6.4. Middleware Support for Performance Improvement

The middleware was used to improve load distribution with the view to reducing outbound communication. It studies the communication pattern of the agents, with whom they communicate most. It will then regroup the agents such that as many peer agents (those that are likely to have frequent communication with each other) as possible are run on the same node, effectively reducing most of the communication traffic to an inbound one. The effectiveness of the middleware was examined with an experiment where agents are initially located arbitrarily without any knowledge of their communication patterns and location of their peers. During the execution, the middleware analyzes the messaging behaviour of the agents, with whom they communicate very often. It then uses the analysis to perform task reallocation by moving as many peer agents as possible to a common node so that most of the inter-agent communication can be internal.

In this experiment, it is assumed that the list of peers an agent communicates with does not change with time. However, it is not necessary

that any two peers have an identical set of peers. Because of this, it is not possible to eliminate outbound communication completely.

If the simulation involves N agents running on M machines, the probability that an agent finds its peers on its own node is 1/M. The number of agents launched per node will be N/M and the rate of outbound communication will then be:

$$OB = (1-1/M)*100\% \quad (1)$$

The middleware can reduce the OB above significantly depending on the number of peers an agent has. The table below shows the reduction in the number of outbound messages as a result of the middleware's intervention. The experiment was run with 500 agents and different inter-agent communication intensities (different number of peers per agent) on a 5-node Grid.

Those runs with larger reduction percentage correspond to cases where agents have relatively fewer peers, so that after reallocation, they find most of their peers on the same node and the communication becomes essentially inbound. If, however, agents interact with too many peers, the middleware may only be able to move a limited number of these peers to the same node and a good portion of the communication remains outbound..

Table 1. Comparison of outbound traffic with and without middleware support.

Run No.	Number of OB Messages		Reduction
	Without middleware	With middleware	
1	1564	458	70.7%
2	5716	3840	32.8%
3	15825	4721	70.2%
4	28691	13393	53.3%
5	32552	8496	73.9%

7. Conclusions and Future Work

The experiment has shown the architectural issues to be considered in designing MABS applications for the Grid and also how a middleware can be used to improve application performance. The findings of this work are useful for platform development, resource planning and simulation modeling. A Grid based platform making use of a performance-aware middleware can support fine tuning of its overlying application if the simulation size, the communication characteristics and the granularity of the application can be determined. Simulation modelers can decide on the architecture of the simulation, complexity of the agent codes, etc., anticipating the performance benefits that can be achieved by designing an efficient simulation.

The messages used in the experiment have small sizes (few bytes). The achieved improvement in performance, i.e., reduction in execution time, however, also depends on the size of the messages. It is therefore necessary to perform additional experiments to determine the performance gain for different sizes of messages, to determine whether the gain in outbound message reduction is translated into a saving in execution time.

We extended the experiment with the previous workload, to build performance models. The model predicts the simulation execution time if it is given the simulation size (number of agents), task granularity, the rate of outbound communication and the number of Grid nodes on which the simulation is executed. An important benefit of the prediction model is that it can facilitate the tasks of Grid resource allocation and scheduling.

The accuracy of the model largely depends on the granularity of the simulation. While execution times for coarse grained simulations (like in figure 4b) are fairly predicted, fine-grained simulations were not accurately estimated. We believe this problem can be partly solved if the model is made comprehensive enough using additional experimental data with fine-grained simulations. However, if the granularity is extremely fine, it will be difficult to perform stable experiments and take accurate measurements due to instrumentation and related overheads.

Although it was attempted to make the compute resources heterogeneous by exposing them to different load conditions, the initial study would have been more comprehensive in an environment with heterogeneity in hardware and software. To address this problem, we are currently expanding our initial experiment setup. Future experiments will be performed on an already

established Globus 4 testbed with around 40 machines over 3 sites in Germany and Sweden.

A synthetic workload with generic features was used to represent a basic MABS application so far in a homogeneous Grid environment. This workload differs from an actual MABS because its model contains several simplifying assumptions. In our current work, we are studying how the middleware can be used in realistic scenarios in a dynamic execution environment consisting of heterogeneous resources. For this purpose, a practical simulation application showing the prominent features of MABS used in social simulation modeling should be empoyed. We thus replaced the workload with an existing MABS application from practice in the transport and logistics domain.

The simulation application focuses on effects of control policies on freight transport chains. It is used as a decision support system for the various stakeholders of a transport chain (customers, buyers, suppliers, production, transport coordinators, etc.). It captures important information such as production capacity, storage, vehicle loading and unloading time, vehicle capacity, speed, environmental performance, and others. The simulation model also incorporates the interaction between all the entities, which are rightly modeled as agents.

A version of the simulator was implemented using the JADE platform on a stand-alone machine. JADE can be used in a way that fits the middleware architecture given in figures 1 and 2. It simplifies the implementation of MABS applications through a middleware that complies with FIPA agent specifications and services such as white-page services, yellow-page services, message transport and parsing services. The JADE platform can also be used across Java-enabled heterogeneous machines on the Grid. It is based on an execution container, which provides a complete run-time and communication environment for agent execution.

Initially, JADE was not developed with simulation applications in mind [16]. The messaging service of JADE also indicates that it was not designed for high performance computing systems. In order to carry out a large-scale simulation on the Grid using JADE, it would be necessary to identify the features of JADE that can be sources of performance bottlenecks.

One cause of performance degradation while implementing the workload as a distributed simulation with JADE is that of inter-agent messaging. We are therefore working on extending JADE to be suitable for the Grid

environment, to efficiently execute communication-intensive simulations. One extension is the decentralization of the messaging service so that inbound messages can be delivered without congesting the network and burdening the services on the main JADE platform. Another enhancement is the adaptation of our middleware for task reallocation and agent migration as explained earlier in section 6.4. We employ decentralized directory facilitators and yellow page services to provide the necessary information for the middleware to make appropriate decisions on task redistribution. The middleware studies the messaging and deployment patterns of peer agents by interacting with the messaging and directory services as the simulation progresses.

8 References

1. Cioffi-Revilla, C.: Invariance and universality in social agent-based simulations. Proc. National Academy of Science USA, 99 (2002) Suppl. 3: 7314-6
2. Davidsson, P., Holmgren J., Kyhlbäck H., Mengistu D., and Persson M. Applications of multi-agent based simulations. Multi-Agent-Based Simulation VII, Lecture Notes in Computer Science, Vol. 4442, Springer, (2007)
3. Sansores, C. and Pavon, J.: A framework for agent based social simulation. The Second European Workshop on Multi-Agent Systems (2004), Barcelona, Spain.
4. Gasser, L., Kakugawa1 K., Chee1 B., and Esteva1 M.: Smooth scaling ahead: Progressive MAS simulation from single PCs to Grids. Multi-Agent and Multi-Agent-Based Simulation, Lecture Notes in Computer Science Vol. 3415, Springer (2005) pp. 1-10.
5. Baker, M., Apon, A., Ferner, C., and Brown, J..: Emerging Grid standards. Computer, Vol. 38 No. 4 IEEE Computer Press, USA, (2005) pp. 43-50.
6. Sotomajor, B.: The Globus toolkit 4 programmer's tutorial. http://gdp.globus.org/gt4-tutorial/. Visited March 2006.
7. Helsinger, A. Lazarus, R., Wright, W., and Zinky, J.:. Tools and techniques for performance measurement of large distributed multi-agent

systems. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, ACM (2003) pp. 843–850.

8. Xu, Z., Miller, B.P. and Naim, O.: Dynamic instrumentation of threaded applications. Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (1999), Georgia USA.
9. Author, N.: The behaviour of Java threads under Linux NPTL www.amitysolutions.com.-au/documents/NPTL_Java_threads.pdf. Visited: July, 2006.
10. Tichy, W.F.: Should computer scientists experiment more?. IEEE Computer, USA. Vol. 31 No. 5 (1998), pp.32-40
11. Timm I.J., Pawlaszczyk D.,: Large Scale Multiagent Simulation on the Grid. Proceedings of 5th IEEE International Symposium on Cluster Computing and the Grid, (2005). IEEE Computer Society, Washington, DC, USA
12. Jarvis, S.A. Spooner, D.P., Keung, H.N.L.C., Nudd, G.R.: Performance prediction and its use in parallel and distributed computing systems. Proceedings of the 17th International Symposium on Parallel and Distributed Processing (2003). IEEE Computer Society, Washington, DC, USA
13. Badia, Rosa M.: Performance Prediction in a Grid environment. 1st European Across Grids Conference, Santiago de Compostela (2003).
14. Puppin D., Tonellotto N., and Laforenza D.: Using Web Services to Run Distributed Numerical Applications. In: D. Krazmuller (Eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface. Lecture Notes in Computer Science, Vol. 3241 Springer, Berlin Heidelberg, pp. 207-214 (2004)
15. Marieto, M.B. et. al.: Requirements Analysis of Agent-Based Simulation Platforms. In: J.S. Sichman, F. Bousquet, P. Davidsson (Eds.): Multi-Agent-Based Simulation. Lecture Notes in Computer Science, Vol. 2581. Springer-Verlag, Berlin Heidelberg (2003)
16. Page, B., and Kreutzer, W.: The Java Simulation Handbook. Shaker Verlag, Aachen Germany (2005)

FOUR

Performance Optimization for Multi-Agent Based Simulation in Grid Environments

Dawit Mengistu and Peter Tröger
Blekinge Institute of Technology
{dawit.mengistu, peter.troger}@bth.se

Abstract

Multi-agent based simulation (MABS) is a discrete event simulation technique used to study complex systems with entities having social and autonomous behavior. MABS applications are characterized by unpredictable execution behavior and high communication-to-computation ratio.

In this paper, we propose an adaptation strategy to support efficient execution of large-scale MABS applications on typical Grid infrastructures. To achieve this objective, the behavior of MABS applications and the execution environment is investigated, in order to constantly obtain performance prediction models. These models will then be used to realize dynamic load balancing and resource allocation schemes.
We discuss our basic approach, initial experimental results, the planned future research and an application of our research in the transportation and logistics simulation domain.

1. Background

The Grid provides the much needed computing resources for large-scale scientific problems. One area that can benefit from the Grid is simulation, in particular multi-agent based simulation (MABS).

MABS is a discrete event simulation (DES) technique used to study complex systems and phenomena consisting of entities having social and autonomous behavior. Researchers in the fields of economics, sociology, biology, and ecology use MABS as a replacement and/or complementary to macro-simulation [1].

The agents in a MABS form the entities of the simulated real world phenomena (such as human beings or other objects of interest). An agent is an autonomous computational entity existing in the form of programs, with sensory and perceptual capabilities, and can make decisions and acts on the environment it is embodied in through effectors [2]. Agents communicate with each other, to influence one another or to pursue collective intentions using standard language constructs.

Developing MABS applications from scratch is a time-consuming task. There exist multi-agent platforms capable of handling the most generic features [3]. They provide a framework with standard templates for modeling generic agent behavior, to manage the launching of the simulation, delivery of messages, scheduling and timing of events, and provide mechanisms for life-cycle management. A number of global communities work in developing standards for multi-agent platforms. The "Foundation for Intelligent Physical Agents" (FIPA) is the prominent forum for agent technology standardization.

In certain social phenomena, partial simulations do not always produce meaningful results. It is thus desired to model and simulate the entire phenomena and implement a large-scale MABS application, which demands huge computational resources. The architectural behavior of MABS poses opportunities and challenges with regards to their implementation. MABS applications are built upon autonomous computational agents that can be conveniently deployed in a distributed execution environment. These agents have a highly unpredictable execution behavior attributed partly to their communication characteristics and inherent constraints of the MABS application domain. Inter-agent communication that takes place between agents is a source of latency and performance degradation. The time management aspect and the need to maintain global synchronization also severely affects application performance.

2. Motivation

The objective of our research is to improve the support for running MABS applications in cluster and Grid environments.

Usually, the agent implementations for the simulated entities are realized as set of threads executing asynchronously. A MABS application is therefore a parallel application that can be executed on parallel architectures. The complexity of such simulation may be determined by:

- The number of agents / threads running;
- The intensity of inter-agent interaction given by the amount of messaging;
- The granularity and complexity of the agents' task implemented as its computational body

Running a simulation on a MABS platform involves the following steps:

1. Setup and initialization of the MABS platform in a hosting environment;
2. Creating simulation agent implementations with the platform supplied templates;
3. Implementing additional features of the agents and the communication behavior;
4. Launching the simulation application.

Existing multi agent platforms do not have built-in mechanisms to implement these steps in the Grid environment in a seamless manner. Some of these platforms also do not address the problems of large scale MABS. Furthermore, in their standard configuration, none of the platforms are optimized for execution in the Grid environment.

In order to achieve the best possible performance in Grids, applications need to be dynamic and adaptable to the changes in the execution environment. Due to this, specific middleware were introduced in the past for the various application domains, e.g. like the Cactus toolkit [10].

3. Research Plan

Large scale MABS are characterized by their high communication-to-computation ratio. In several MABS applications, there is also a need for maintaining global time synchronization across all execution nodes. Such applications are not generally suited for execution in the Grid environment with its resource heterogeneity, load fluctuations, constraints and policies at

execution nodes, etc. However, by making necessary alterations on the run-time behavior of MABS applications, a considerable improvement in performance might be achievable. Possible solutions to this problem include:

- Use of appropriate deployment and load reallocation strategies such as agent migration to reduce cross-node inter-agent communication and to correct load imbalance;
- Use of aggregation techniques for delivery of cross-node inter-agent messages;
- Optimizing the application's threading level;
- Enforcement of an optimal time synchronization method at the global level.

In our research, we therefore study how a MABS platform can be adapted for an improved performance in the Grid environment. We investigate the behavior of the MABS application itself and the execution environment at runtime. The resource requirements of running MABS applications are recorded and analyzed in order to allow a performance prediction based on the analyzed data to devise a better allocation and scheduling strategy and accordingly initiate suspension, migration, and termination of agents.

We shall investigate two types of approaches on how to realize the proposed adaptation components for the MABS application:

- Centralized approach: The relevant performance data is collected from the execution nodes and stored in a central database for analysis purposes. This data includes cross node traffic data, agent task granularity, load information on execution node, etc. This data is then processed centrally to figure out the interaction topology and peer-list of agents to propose reallocation and bundling strategies using heuristics algorithms. The major limitation of this approach is that with growing simulation size and increasing number of Grid resources participating in executing the MABS application, the data collection and analysis tasks may be overwhelming. The resulting computational overhead can even make the viability of the solution for large MABS questionable.

- Self-configuration approach: an optimal (or sub optimal) distribution of agents with minimal cross node communication can be achieved using adaptive algorithms locally. It allows optimal threading level and load

distribution at the nodes. The decisions for changes in the application configuration will be made at relevant local levels with minimal central interference. Though this approach may not always yield optimal solutions, it gives faster results with minimal computational and communication overheads. This approach requires implementing the data instrumentation and analysis components in a decentralized manner.

4. Initial Approach and Experiments

In order to study the general applicability of wide-area distributed processing in our problem domain, a generic MABS application was implemented for initial experiments in a Globus-based testbed. We used the WSRF programming model in order to ease up the development. Each Worker Grid service in the simulation hosts an equal number of agents, instantiated as user threads. The simulation is therefore run as a parallel application launched from a master (client) node where the workload is divided into balanced loads and distributed accordingly to the Grid services. At this stage, only simple agents with basic functionalities were used.
The experiment is designed in such a way that during one simulation cycle, each agent sends out a message after a fixed amount of computing. This cycle of computation-communication runs multiple times.
The following parameters are used as control variables in the initial study of the resource allocation problem:

- Number of agents, N
- Outbound communication rate (OB in %), the ratio of the number of messages to other nodes to the total number of messages sent out by an agent
- Granularity (G in ms), the amount of computational work the agent performs before sending a message
- Number of computational nodes (M)

Measurements were taken by varying the above parameters in a controlled environment. From the measured data, performance prediction models were built to compute the execution times T_{st} and T_{Grid} of simulation runs in stand-alone and Grid environments respectively, as functions of the control parameters:

$$T_{st} = T_{setup} + K \cdot h(N,G) \qquad (1a)$$

$$T_{grid} = T_{setup} + K \cdot f(N,M,G,OB) \qquad (1b)$$

h and *f* are functions that estimate the effects of the parameters in stand-alone *(N,G)* and Grid *(N,M,G,OB)* environments respectively. In a stand-alone simulation, *M=1* and can be omitted; since all agents reside on one machine, all communications are inbound and *OB* is not a relevant variable.

T_{setup} is the time required to fully deploy and launch the simulation. This time depends on the number of agents involved in the simulation and should be studied separately, since it may even be longer than the useful simulation run-time.

K is a proportionality factor representing the length of the simulation if the execution events are assumed to be independent. Obviously, *K* has no effect on the set up time.

Since the simulation size is usually given in terms of the population size of the simulated entities, which is the number of agents, it is also possible to write the execution time as a function of *N* (the Grid version is shown here, the stand-alone has also a similar form):

$$T_{grid}(N) = T_{setup} + a_m N^m + a_{m-1} N^{m-1} + \ldots + a_1 N + a_0 \qquad (2)$$

Where a_0, a_1, …, a_m are coefficients dependent on the Grid simulation environment and the architecture of the application. They are given by:

$$a_i = f_i(M, G, OB) \qquad (3)$$

f_i approximates the combined effect of the three parameters.

To determine the coefficients, data is collected from repeated runs of the simulation for different values of the input parameters. The measurement data were analyzed using the MATLAB software for the initial experiment, in order to build the required performance models. In future, this computation will be part of the performance modeler of the proposed solution.

Beside the computational load, there is also a relationship of simulation performance to the amount of communication between agents. In parallel applications, one technique to reduce the cost of communications is to minimize outbound messaging. MABS, however, have inherent limitations to

employ this technique directly. It is assumed that the list of peers an agent communicates with does not change in time. However, any two peer agents may not necessarily have identical sets of peers between themselves. Because of this, it is not possible to eliminate outbound communication completely.

Our proposed solution tackles this issue by rearranging highly interacting agents on the same node. After an initial random placement, messaging patterns are observed through the monitoring services. Based on this data, a heuristic clustering algorithm identifies groups of peer agents that can be co-allocated in the further execution:

1. Starting with any arbitrary Grid node q, find the number of agents n_q, running on it.
2. For each agent a_i running on node q, create a set S_i of peer agents it communicates with
3. From S_i, identify agents ($a_0, a_1, ..., a_{i-1}, a_{i+1}, ... a_k$) executed on the same node as a_i and repeat step 1 for each, create $S_0, S_1, ... S_{i-1}, S_{i+1}, ... S_k$.
4. From the sets $S_0, S_1, ..., S_{i-1}, S_i, S_{i+1}, ... S_k$., find a common set of peer agents, S_p. Agents belonging to S_i and S_p are peers and can be located on the same node. If the number of peers obtained in this way exceeds the capacity of the node n_q, the excess can form another peer set to be located in the remaining nodes in orderly fashion.
5. Repeat this process until all agents are grouped as above.
6. Reallocate the agents in accordance with the new grouping such that as many peers as possible are deployed on the same node.

As a further effort to relieve the network congestion, messages destined to a node are aggregated together for dispatching. Similarly, messages leaving a node are likewise collected together and dispatched. The messaging agent usually delivers or receives several bundles of messages when it takes control of the CPU.

For time-driven MABS, all agents advance execution at a synchronized pace of simulation time steps to imitate real time. Agents that have completed a one time-step task remain in a waiting state until all agents get to the same level.

4.1. Initial Results

The experiment was run with the following range of input variables:

M = 1, 2, 3, 4, 5, 6
OB = 0, 10, 20, ..., 90, 100
G = 0.2, 1, 2, 10, 20, 100, 200 ms
N = 50, 100, 150, ..., 3000

The resulting performance model in the initial experiment for a granularity G=10ms is shown in equation 4.

$$T(N) = ((-0.0024*M) + (0.0001*OB) + 0.0113) * N^2) + ((-0.2748*M) + (0.020*OB) + 1.1535)*N) + ((75.2585*M) + (2.7441 * OB) - 113.757) \quad (4)$$

(4)
(5)

The thread setup time (in seconds) is given by the following model, valid for 100<N<1000:

$$T_{setup} = (0.000476 * N^2) - (0.03752 * N) + 8.2625 \quad (5)$$

The coefficients of OB are small indicating that the effect of outbound communication is minimal for large N (larger simulations).

The models are validated with measurements as can be seen in the data shown in table 1. The chosen values of the inputs are selected from the model's valid ranges.

G (ms)	M	OB (%)	Response Time (s)		Dev. (%)
			Predicted	Measured	
2	2	10	616	483	22%
2	4	10	660	751	14%
2	5	10	682	790	16%
5	2	20	373	320	14%
5	4	20	353	358	1%
5	5	20	405	403	0%
10	2	30	330	368	12%
10	4	30	256	245	4%
10	5	30	292	281	4%

Table 1. Performance prediction using N=180

If there is large outbound communication, scalability obviously cannot be achieved, since raising the number of worker nodes only increases the communication cost much more than the saving in computation time.

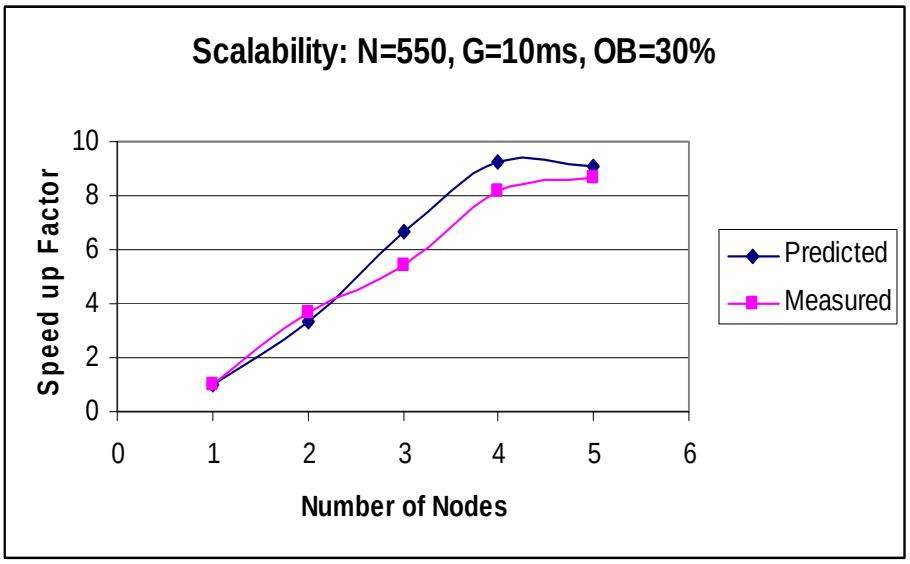


Figure 1: Scalability for increasing node count

If the simulation involves *N* agents randomly distributed on M machines, the probability that an agent finds its peers on its own node is *1/M*. The number of agents launched per node will be N/M and the rate of outbound communication will then be:

$$OB = (1-1/M)*100\% \qquad (6)$$

Using the earlier mentioned algorithm, the OB can be reduced significantly depending on the number of peers an agent has. Table 2 shows the reduction in the number of outbound messages as a result of this intervention. The experiment was run with 500 agents and different inter-agent communication intensities (different number of peers per agent) on 5 participating machines.

Those runs with larger reduction percentage correspond to cases where agents have relatively fewer peers, so that after reallocation, they find most of their peers on the same node and the communication becomes essentially inbound. If, however, agents interact with too many peers, it may only be possible to move a limited number of these peers to the same node and a good portion of the communication remains outbound.

Run No.	**No. of OB Messages**		**Reduction**
	Without support	With support	
1	1564	458	70.7%
2	5716	3840	32.8%
3	15825	4721	70.2%
4	28691	13393	53.3%
5	32552	8496	73.9%

Table 2: Comparison of outbound messages with and without agent reallocation support

4.2. Related Work and Use Cases

The Caribbean system [6] is a massively multi-agent platform for use in large-scale navigation system of humans. As most agent simulations, this tool is developed as a standalone threaded application, which does not scale up to a distributed parallel simulation environment. Another example is from Shibuya et al. [7], who propose a framework for multi-agent based computational modeling and quantification of social behavior in mobile systems.

Gunderson and Petersen [8] propose a multi-agent based problem solving approach in mobile environments. It mainly focuses on dynamic planning issues of applications running on mobile devices. The work is at a prototype stage and no empirical validations have been produced.

Poggi et al. [9] discuss the difference between agents as enablers or customers of grid capabilities. Like in our approach, they propose the extension of JADE for direct usage in Grid environments, but mainly for rule-based creation and composition of tasks. There is no specific relation to application scenarios or particular grid environments.

Ludwig et al. [11] propose the usage of agents for implementing SLA negotiation facilities in Service Grid environments. Their work mainly introduces a negotiation protocol based on the FIPA standards.

Newman et al. [12] describe agent-based distribution services for data grids in high-energy physics. There work is based on the JINI platform, using a self-organizing scheduling approach. Even though the application domain differs, there scheduling approaches might be interesting for further investigation.

Moreau [13] describes the usage of agent technology for building grid infrastructures. He describes grid environments in terms of a service market place, where agents could act on behalf of both service consumers and resource providers. In contrast to this work, our scenarios focus on agents as grid application.

Transportation Scenario

We plan to use the “Java Agent DEvelopment Framework” (JADE) toolkit as our MABS platform, a widely used multi agent platform based on FIPA specifications. JADE simplifies the implementation of MABS applications

through a layer that complies with FIPA agent specifications and services such as white-page services, yellow-page services, message transport and parsing services. The JADE platform can be used across Java-enabled heterogeneous machines. It is based on an own execution container, which provides a complete run-time and communication environment for agent execution.

The MABS application employed as a use case focuses on effects of control policies on freight transport chains. It is known that besides the positive impact freight transport has on the economy, it has also several drawbacks, such as emission of dioxides, traffic congestion, or accidents [4]. The objective is to study the impact of introducing new government control policies on freight transport chains through a micro-level simulation technique. The simulation model can also be used as a decision support system for the various stakeholders of a transport chain, such as customers, buyers, suppliers, production, or transport coordinators. It captures important information such as production capacity, storage, vehicle loading and unloading time, vehicle capacity, speed, environmental performance, and others. The model also incorporates the interaction between all the entities, which are rightly modeled as agents.

An available version of the simulator, known as TAPAS, is implemented using the JADE platform for stand-alone machines [5]. It was observed that for performance reasons, it is not possible to carry out a full-scale simulation. This observation provides the foundation for the overall research topic. Since the existing simulation is based on JADE, we extend it in order to rely on our improved MABS support for agent placement and reallocation decisions.

5. Conclusion and Next Steps

An important issue in parallel simulation is time management, preserving temporal characteristics of the simulated phenomena. The ordering and causality of events has to be correctly captured and enforced during the execution of the simulation to reproduce the behavior of the real world problem and to ensure repeatability of the experiment. Preserving temporal consistency of MABS applications ported to the Grid environment needs to be explored in detail and appropriate measures are required to adapt existing methods for management and synchronization of simulation time.

In the first experiments, we used a synthetic workload with generic features to represent a basic MABS application. This workload differs from an actual MABS workload because its model contains several simplifying assumptions. Prominent features of MABS applications used in social simulation modeling were not considered until now. A further simplification in the workload was the avoidance of any I/O related activities.

Although it was attempted to make the compute resources heterogeneous by exposing them to different load conditions, the initial study would have been more comprehensive in an environment with heterogeneity in hardware and software. So far, the experiment environment was only a local cluster system of homogeneous nodes. To address this problem, we are currently expanding our initial experiment setup. Future experiments will be performed on an already established Globus 4 testbed with around 40 machines over 3 sites in Germany and Sweden.

We will also replace the initial agent simulation by an existing MABS application from practice in the transport and logistics domain.

We believe that the outcome of this research will be useful for development of large-scale MABS applications on the Grid. Although the work is domain specific, some of the important findings can also be used in other types of parallel distribution simulation applications.

6. References

[1] Sansores, C. and Pavon, J. "A framework for agent based social simulation". *The Second European Workshop on Multi-Agent Systems*. Barcelona, Spain. 2006

[2] Conte, R., Gilbert, N., and Sichman, S. "MAS and social simulation: A suitable commitment." *LNCS V1532, Springer Berlin/Heidelberg.* 1998. pp. 1-9

[3] Kota, R., Bansal, V., and Karlapalem K. "System issues in crowd simulation using massively multi-agent systems" *Proceedings of International Student Workshop on Agents.* Kyoto, Japan. 2006.

[4] Ramstedt, L. "Analyzing the effects of government control policies in transport chains using micro-level simulation", *Licentiate Thesis, Blekinge Institute of Technology*. Ronneby, Sweden. 2005.

[5] Holmgren, J. et. al. “An Agent Based Simulator for Production and Transportation of Products”, *11th World Conference on Transport Research*. Berkeley, USA. 2007

[6] Nakajima, Y. et.al. "Caribbean/Q: A Massively Multi-Agent Platform with Scenario Description Language". *Second International Conference on Semantics, Knowledge, and Grid*. 2006.

[7] Shibuya, K. “A framework of multi-agent-based modeling, simulation, and computational assistance in an ubiquitous environment” *SIMULATION, Vol. 80*. No.7-8. The Society for Modeling and Simulation International 2004, pp 367-80

[8] Gunderson, O.E., and Petersen A..K. “Multiagent Based Problem-solving in a Mobile Environment” *Norsk Informati-kkonferanse.* Bergen, Norway. 2005.

[9] Poggi, A.; Tomaiuolo, M.; Turci, P., "Extending JADE for agent grid applications," *WET ICE.* 2004. pp. 352-357.

[10] Allen, G.; Benger,W.; Dramlitsch, T.; Goodale, T.; Hege, H.-C.; Lanfermann, G.; Merzky, A.; Radke, T; Seidel, E; Shalf, J.: “Cactus Tools for Grid Applications”. *Cluster Computing 4 (2001), Nr. 3*. pp. 179–188

[11] Ludwig A.; Braun P.; Kowalczyk R.; Franczyk B. (2005). “A Framework for Automated Negotiation of Service Level Agreements in Service Grids”. *Proc. 3rd International Conference on Business Process Management (BPM 2004)*. Nancy, France. September 5th, 2005.

[12] Newman H.; Legrand I.; Bunn J.; “A Distributed Agent-based Architecture for Dynamic Services”. *Computing in High Energy and Nuclear Physics.* Beijing, China. 2001.

[13] Moreau, L.; “Agents for the Grid: A Comparison with Web Services” *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. 2002. p. 220

FIVE

Scalability in Distributed Multi-Agent Based Simulations: The JADE Case

Dawit Mengistu, Peter Tröger, Lars Lundberg, Paul Davidsson

Blekinge Institute of Technology
Soft Center, Box 520, 372 25 Ronneby, Sweden
{dawit.mengistu, peter.troger, lars.lundberg, paul.davidsson }@bth.se

Abstract

Agent platforms provide a framework for development and execution of parallel applications such as multi-agent based simulation (MABS). However, these platforms have limitations to support large-scale MABS applications in practice.

This paper aims at investigating and improving the performance of an agent platform with a MABS workload in distributed environments. We carried out an experimental study with the JADE framework as our agent platform. The experiments show the performance characteristics of the workload and that JADE does not scale well due to message transport and agent directory service limitations. We propose solutions to overcome these performance bottlenecks and facilitate the efficient execution of MABS in a distributed environment. Initial experimental results demonstrate the feasibility of the proposed approaches

1. Introduction

Multi-agent based simulation (MABS) is a useful approach to study and understand the dynamics of social systems made up of a large number of interacting entities. It provides a good insight into the evolution of emergent behaviour from the interaction. Research has shown that MABS is a promising technology in modeling social dynamics in biological, social, economic, etc., which often cannot be adequately captured using traditional simulation methods [1].

In some real world problems, it is desired to simulate the entire system since partial simulations do not help understand the system adequately. Such simulations which would involve the modeling of thousands (even millions) of entities as software agents, cannot be executed on a single computer due to resource limitations. Resource rich systems, such as clusters and the Grid, can provide the much needed computational infrastructure to deploy MABS as a distributed application. However, MABS have generally a high communication-to-computation ratio and are thus not considered to be good candidates as workloads for a distributed computing environment. Other MABS implementation features, such as massive multi-threading and the time synchronization required to maintain event causality, are additional factors that negatively affect the performance of a distributed simulation [10].

MABS applications can be developed from scratch as stand-alone software using any programming language. However, the most efficient way is often to implement them as applications deployed on a multi-agent platform, a middleware layer between the application and the execution environment. A multi-agent platform offers functionalities that provide standard APIs for agent management, communication and information services. There exist a number of multi-agent toolkits specifically developed for MABS. However, these toolkits are often intended to model only specific social simulation scenarios and cannot serve as general purpose platforms for distributed simulation. One approach is therefore to customize generic multi-agent platforms to support a wider class of simulation applications. The problem with this approach is that, because the platforms were not originally designed for MABS, they are not optimized to execute large scale MABS applications.

In this paper, we study the performance of a multi-agent middleware and propose ways to improve its execution characteristics. We use the Java Agent

Development Framework (JADE) to conduct our experiments. JADE was chosen because of its rich set of APIs and its wide-spread usage.

The rest of this paper is organized as follows: After a short overview of multi-agent platform principles, the next section gives a brief outline of our approach. A description of the experimental setup is then presented followed by the results. We discuss the important findings of this experiment and conclude the paper with our intended future work.

2. Overview of Multi-agent platforms

Multi-agent platforms facilitate the development of MABS applications through standard APIs which support execution control, agent communication, life-cycle management, and information directory services. Recommendations on the architecture of multi-agent platforms are made by different bodies, among which those from the FIPA (Foundation for Intelligent Physical Agents) are the widely used ones [2]. FIPA provides standard specifications for services which are used as foundation for the architecture of an agent-based implementation. Some platforms provide only high level interfaces for application developers, while others give access to low level implementations.

Multi-agent platforms were originally developed to support agent based applications in domains such as online trading, computer games, defense applications, transportation and logistics, supply chain management, etc. These applications use relatively few agents which typically have a large memory foot print. In applications where the business logic demands distributed deployment of the agents, little or no global control of the overall execution is needed. Even if the agents pursue their high level objectives in collaboration with others, the processing of their program code largely takes place asynchronously and independently of other agents.

In contrast to traditional multi-agent domains, large-scale MABS applications are implemented with numerous light-weight agents typically characterized by intensive inter-agent communication and strict time synchronization. Although agent platforms provide the necessary programming and implementation models for MABS, they do not satisfy some of its requirements [3]. The limitations include:

1. Most platforms lack a mechanism to manage simulation time and properly guide the execution of the simulation in synchronized lock steps.

2. The platforms do not have a mechanism to automate the generation and deployment of agent models for a simulation scenario. This problem is particularly serious if the simulation involves thousands of agents that have to be manually encoded by users.

3. The agents, being typically modeled as heavy-weight objects, pose performance problems when the simulation involves a large number of agents. In addition, such platforms expose only high level functionalities to developers, restricting developers from writing computationally efficient codes.

4. Many platforms do not have a visualization module to present results of simulations having both spatial and temporal behavior. As the main intent of using MABS is to study emergent and evolutionary phenomena, users are interested in visualizing how a given phenomena evolves in time and space.

The lack of these important features puts an additional programming burden on MABS developers.

3. Approach

Achieving optimal performance for large-scale MABS applications in a distributed environment is not trivial. As described earlier, the scope of our work is to use the JADE agent framework as a platform for multi-agent based simulation, to experimentally investigate its performance characteristics using a MABS workload and, propose solutions to achieve performance improvement.

The first task is to understand the architecture of the JADE environment taking its intended purpose into account. This requires an understanding of JADE's architecture, agent deployment and execution model, and adapting it to our case.

The next step is the creation of a representative workload that contains the essential features of a distributed MABS application. This requires modeling agent roles as computational tasks, defining agent interactions through inter-thread communications, and, maintaining event causality by implementing

thread synchronization. From the workload model, relevant performance parameters are identified. The parameters can be considered in performance prediction models for general MABS use cases.

With the relevant application parameters such as the communication-to-computation ratio at hand, we propose a set of optimizations for the JADE environment and evaluate their effectiveness.

4. Experiment Description

4.1. JADE as simulation environment

JADE implements the basic agent platform services defined in FIPA specifications. These services themselves are implemented as agents: The agent management system (AMS) for life cycle management, the directory facilitator (DF) for agent directory look up services (white page and yellow page), and the agent communication channel (ACC) for message transport [5].

The JADE run-time environment is a Java container that hosts user created agents. It is launched in a Java virtual machine and provides access to the basic agent services through a set of APIs.

In a distributed JADE environment, containers are launched on several hosts. The central platform services reside on the first launched container called the main container. All other containers need to register themselves with the main container at start up.

Each JADE agent is registered on the main container with a unique global address. It runs in its own container as an individual thread. JADE employs the LDAP protocol for the agent directory service.

Agents can send messages to one another based on the standardized agent communication language (ACL). If a message is sent between agents in the same JADE container, the Java event passing is used for message delivery. For communication between agents living in different containers RMI is used [15].

In the case of MABS, the agent program code implements the task of the simulated entity, the communication and the time synchronization activities.

The simulation agents are launched on all containers except on the main container, which is reserved for basic agent services.

Simulation time advance is controlled at two levels. When all agents in a container finish their task for the current time, a *sync* signal is set on the host and the main container is notified. When *sync* signals from all hosts arrive, the main container issues a global *sync* reply signal to all hosts notifying them to start the next time step. During the synchronization period, the agents will have to remain in a wait state.

When a platform is initialized for distributed simulation, worker nodes need to know the IP address or host name of the main JADE container to register themselves in the platform. This approach is suitable for execution on machines whose addresses are known before the simulation is launched. However, in distributed computing environments such as the Grid, where users cannot normally know the machine(s) on which their application is to be executed, it will not be possible to run a distributed simulation. To overcome this problem, we implemented an approach where the worker nodes determine the address of the machine holding the main platform container during run-time on their own.

4.2. Workload Design

Due to the wide range of real world problems they model, the architecture of MABS applications is diverse. We therefore modeled our workload based on a very generic parameterized simulation application.

The workload consists of simulation agents that are launched on different computers. They perform some computational task within a simulation time-step and then synchronize with one another at the end of the time-step. To keep the workload model less complex, the agents are deployed over the available nodes in equal numbers. They are also assumed to have identical role and execute the same code during their life time.

The progress of the execution is monitored by the number of simulation steps executed. During one lock step, each agent executes its computational task and sends out messages to its peers, which could be on the same node or on another node. It then waits for a reply, on its way sending replies to messages directed to it, if any. If the agents have their peers deployed on the same node, the messages are internal or inbound, while if they are on a different node, the messages will be outbound. The ratio of outbound

messages to the total number of messages is an independent variable of interest, since it allows studying network latency effects in the MABS execution.

The sequence diagram for the execution of one simulation step is presented in figure 1. The length of the vertical boxes corresponds to the computational granularity, the time it takes an agent to execute the code defining its task during one simulation step.

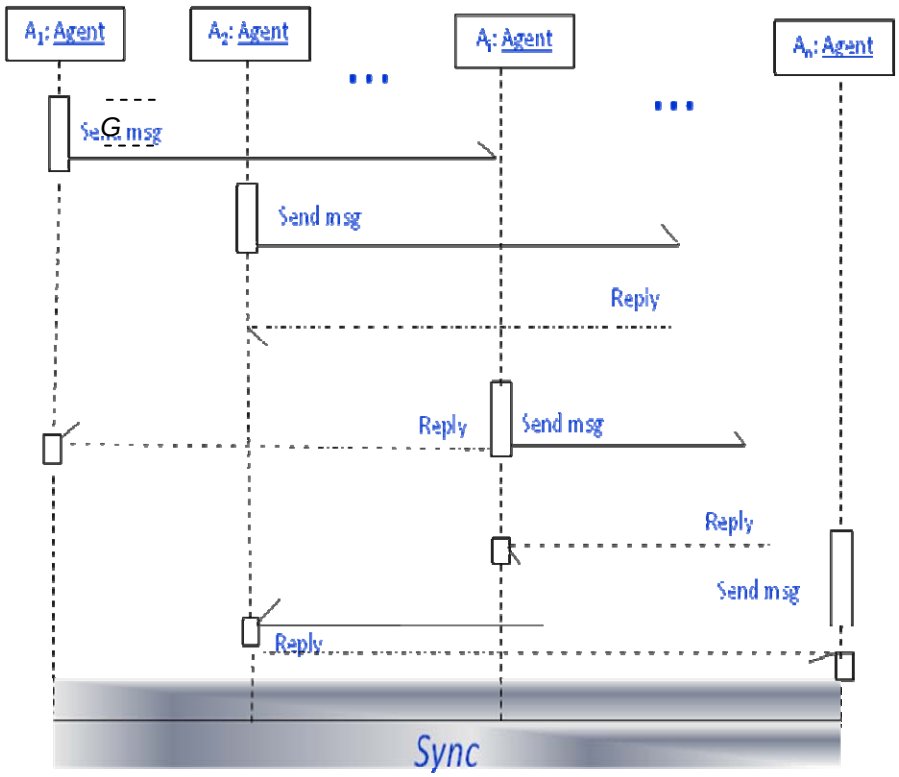


Figure 1. MABS Communication Pattern

In this experiment, the task granularity denoted by *G* is assumed to be the same for all agents. The granularity of the agent task is a measure of the

amount of computation an agent executes before entering the communication phase of one simulation time step. If the duration of one simulation time step is T, considering context switching to be small compared with G, and if n agents execute their tasks during one simulation step, the processing time T_p is given by

$$T_p = nG \qquad (1)$$

The difference between T and T_p is the idle time the agents spend on waiting for replies and the arrival of the global *sync* signal. The communication-to-compu-tation ratio R is therefore given by

$$R = (T - T_p)/ T_p \qquad (2)$$

A small fractional value of R indicates that the application spends less idle time in waiting during the communication and synchronization phases. The objective of this study is essentially to understand the factors affecting R and how to reduce it. To achieve this, we used the following parameters which are also the independent variables of the experiment:

- Simulation size given by the number of agents N
- Task granularity in milliseconds G
- Percentage of outbound messages OB
- Number of machines (compute nodes) M

We ran the simulation by varying one of the variables (N, G, OB, M) at a time to understand the influence of each parameter on performance. The following measurements were taken for each case:

- Mean time to complete one simulation time step
- Yellow page request/response time
- Local synchronization time
- Global synchronization time

5. Results

The experiment was conducted on a Linux based cluster system with 1GHZ single core processors. The head node which hosts the main container has 1GB RAM while the workers which host the other containers have 512MB. JADE version 3.5 is installed on all nodes and the containers hosting the agents are deployed on java virtual machine (JVM) version 1.5.

In each experiment, the workload is executed for several simulation steps. The measurement data presented here show the mean values of the respective quantities over the execution period. Measurements from the first few simulation steps during initialization and warm up are discarded and thus not included in calculation of the mean.

5.1. Communication-to-computation ratio

If the agent tasks are fine grained, the time spent on executing the main agent code with in a time step will be obviously much smaller than that of the communication time. Figure 2 shows the according communication-to-computation ratio (R) for different granularity rates with the outbound communication rate as an additional parameter.

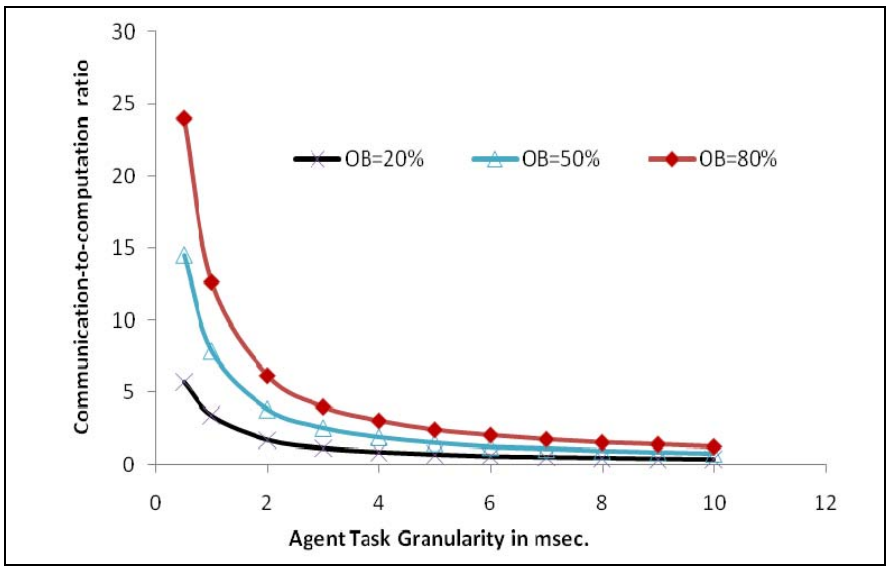


Figure 2. Effect of granularity

In this experiment, the workload contained 2000 agents deployed on four worker nodes of the cluster. The processor utilization rate is very low if the application granularity is small, since most messages are sent between agents living on different nodes.

5.2. Effect of synchronization

To measure the effect of local (node level) and global synchronization, the workload was modified. In each time step, the agents do not perform any computational task, nor do they communicate with each other. Instead, when an agent thread gets control of the CPU, it immediately yields control and waits for the *sync* signal. This process is repeated over several time steps. The elapsed time in this case is then due to context switching and synchronization only. The application was run on the cluster on four worker nodes for different simulation sizes (N=400…6000)

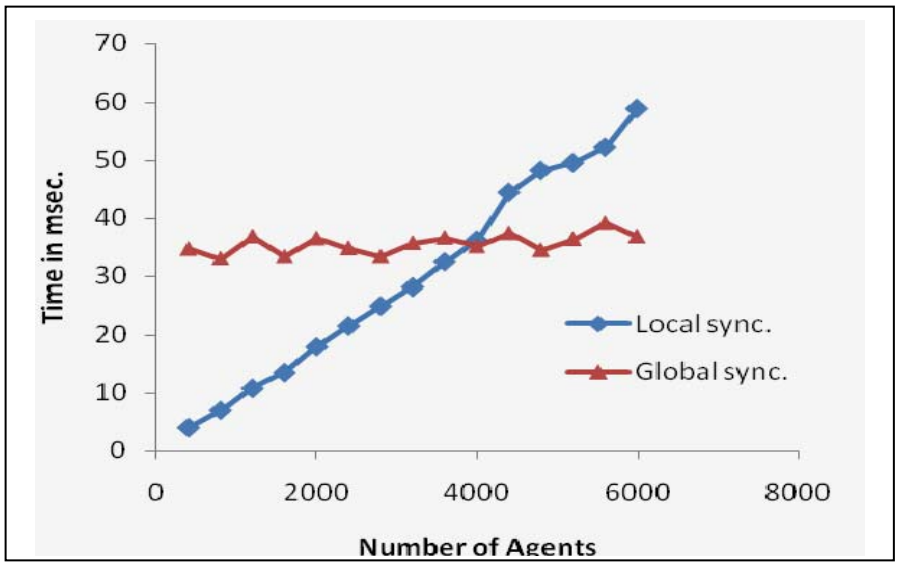


Figure 3. Local and global synchronization times

The experiment results show that there is a break-even point, where the high number of parallel activities on one node leads to serious synchronization problems with shared data structures. The impact of network

latency on global synchronization, and hence on the overall performance of the application then becomes no longer significant. The maximum number of agents deployed in a container was limited by the maximum number of threads creatable by one Java process.

5.3. Optimization of JADE

From initial experiments, we observed that the yellow page response time increases lineary with the number of agents in the simulation. It was further observed that message delivery time was also affected in a similar manner.

We identified that the agent directory service used by JADE is not suitable for large scale simulations as our workload. This is due to the use of LDAP, a protocol which does not scale well for a large number of concurrent requests [13]. JADE's communication service, although is efficient for a workload with minimal communication needs, also becomes a source of performance loss for large scale simulations.

For large simulation sizes, the JADE messaging service adds a significant overhead to the overall execution time. This is true even if communications are entirely taking place between agents residing on the same node. Due to the identification of the above performance bottlenecks, we implemented optimization strategies:

1. Yellow page (and white page) services.

We introduced a tailored directory replication technique, where directory information is mirrored from the main container to others. Agents need to query into their local host directory cache rather than congesting the central directory service. The information cached in the local directories is assumed to be valid during one simulation step: no agent is created, killed, migrated, or changes its registered behavior during this time. This assumption reflects our experience from the MABS application domain. The replicated data will be obsolete if a change in the status of agents occurs during the simulation. Accordingly, we integrated mechanisms to dynamically refresh the contents of the local copies, based on the identification of write requests on the master directory copy.

2. Message transport services.

We also optimized the inbound communication by replacing JADE's native messaging service with our proposed solution. When agents send messages to peers on the same node, delivery is handled through newly added data structures, while outbound messages are still sent and received through the native service.

These optimizations brought significant improve-ment in the performance of our simulation application, as it can be seen in the following sections.

5.3.1. Message Delivery.

Optimizing the messaging service for local message delivery can improve the performance significantly as can be seen in Figure 4. The figure shows a comparison of execution times for one simulation step with native JADE messaging service and our modification. To demonstrate this effect, the application was run on a single node so that all communication is local (inbound).

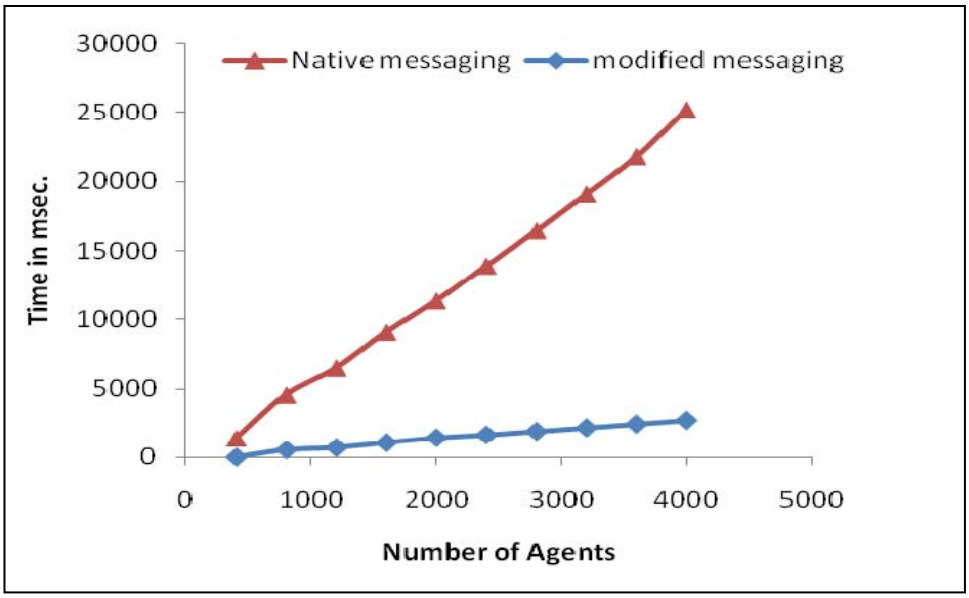


Figure 4. Optimizing the local message delivery

5.3.2. Directory service

Directory replication significantly reduces the network traffic generated by individual agents requesting information from the service.

A static directory cache may not be up to date if frequent changes take place in the simulation configuration such as the addition, removal or migration of agents, changes in the role of individual agents, etc. A dynamic caching strategy, where the directory information on local nodes is refreshed at the end of every time step of the simulation was also employed. This slows down the execution, but is still much faster than the non cached naïve directory request approach.

Figure 5 shows a comparison of mean simulation times per time step for three cases. Both the static and the dynamic cache approach scale well with increasing number of agents in the simulation application.

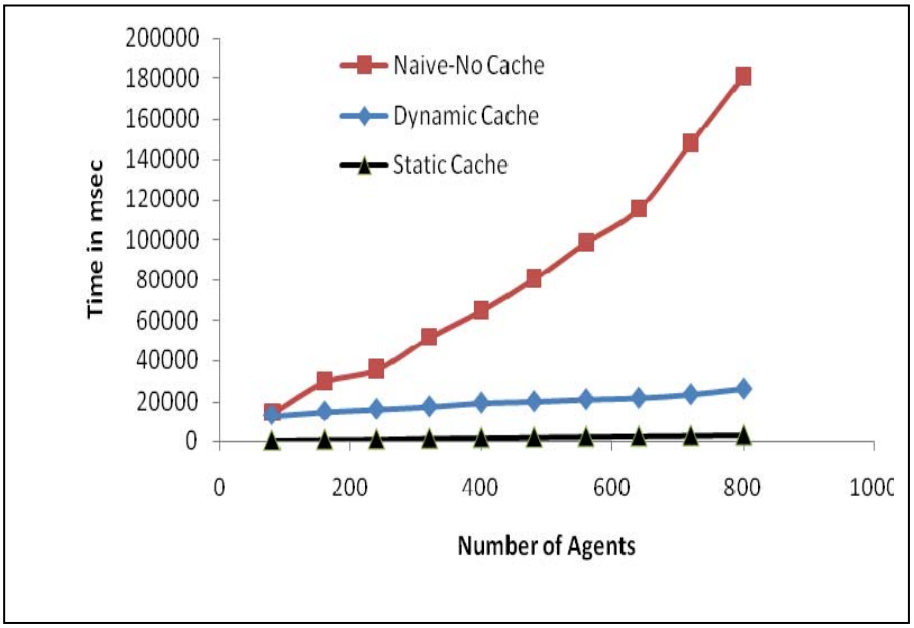


Figure 5. Effect of directory replication.

6. Discussion

Based on our experience with an extreme agent-based simulation example, we conclude that the functionalities of the JADE framework meet the basic requirements of MABS. However, it does not scale well for simulation sizes involving a large number of agents. The major reason for this is the inefficiency of the JADE agent directory service. Because this service is used frequently by the other platform services, its inefficiency affects other services too.

In large-scale MABS, inter-agent communication may cause substantial delays. When a message is delivered, the JADE message transport service needs to know the receiver agent's status (whether it is active or dead) and address (if it is on the same node or not) by accessing the directory service every time. Since the default directory service which employs LDAP has a slow response behaviour, it is overwhelmed by a large number of concurrent requests.

The improvement brought in our work is based on the premise that access to the directory service should be kept to the minimum possible. Efficiency in local messaging (Section 5.3.1) was achieved by bypassing the directory service for local receiver addresses. The caching of directory information (Section 5.3.2) also significantly reduces the number of concurrent requests sent to the central directory service. Our ongoing work investigates the possibility of using an existing distributed shared memory (DSM) system, in order to improve the performance of the directory replication process.

As with other parallel applications, finely grained MABS with high rate of outbound communication have poor performance. However, the following changes in the implementation and architecture of the application can result in an improvement:

1. Reduction of outbound communication: To minimize the number of network requests, outbound messages can be aggregated before they leave the originating host. Instead of making RMI calls for every message, it is easier to store the messages in a buffer for bulk dispatch. Studies show that this can be an effective approach for simulations executed in a distributed environment such as clusters [11].

2. Agent redistribution: If the communication pattern of the agents is known, it may be beneficial to redeploy the agents in such a way that peers are placed on the same node to the extent possible. This involves the migration of agents

from their current location to other nodes. Although this has the direct effect of reducing outbound communication, the overhead associated with migrating agents has to be weighed in. Agent migration involves the transfer of an agent's code and its state information to a new location. Experiments show that migration on JADE is an expensive operation, particularly if the number of agents to be migrated is high [14]. Our future work therefore also investigates migration decisions based on performance prediction models.

3. Increasing task granularity: If the simulation problem allows the bundling of several agents into one thread, it will be possible to manage communication activities at a coarser level. It also minimizes context switching overheads.

7. Related Work

There are several studies done on performance and scalability with agent platforms in a distributed environment. However, only few of these address MABS as an application domain and its specific requirements.

The communication performance of the JADE framework was studied by Vitaglione et. al [15] and Chmiel et.al [18]. These studies show the behaviour of JADE message transport system and the underlying RMI implementation under different load conditions. They also explain the optimization strategy used in JADE to distinguish between inbound and outbound communication. However, the studies do not explain the bottlenecks of the messaging mechanism and thus did not look at ways of improving communication performance.

Wang, Turner and Wang [19] proposed an approach to integrate JADE simulation agents with the high level architecture (HLA), a framework designed for simulation interoperability and coordination of decentralized components to enable collaborative simulation. Their study deals with communication between agents and synchronization of messaging to maintain causality in a distributed environment. The approach is applicable in collaborative simulation scenarios and has therefore less relevance to use cases involving the envisaged MABS workloads that are launched and controlled centrally. Besides, it does not consider performance and scalability issues that arise with large scale simulations.

Theodoropoulos et. al [20] presented a middleware called HLA_Grid_RePast for large-scale agent based simulation. As the name suggests, this middleware is deployed in the Grid environment based on the Repast agent platform. It also uses HLA to combine multiple components into a collaborative simulation. Although this work recognizes the importance of communication efficiency for the overall performance of the simulation, it does not look into ways of reducing communication overheads. The RePast platform used as the foundation of this work offers only high level interfaces and thus programmers have limited control on the implementation of the underlying agent message transport.

8. Conclusion and Future Work

This work demonstrated how the JADE multi-agent framework can be tailored to support large scale MABS in a distributed environment. JADE offers a friendly agent development environment, transparent interfaces and high-level message transport and directory services. This flexibility enabled us to elaborate novel approaches to make JADE a scalable platform addressing performance requirements of MABS applications.

Our findings should be helpful for MABS developers to make informed design decisions and write efficient simulation applications. The results obtained so far are also necessary to build performance prediction models, which is part of our future work. We will study JADE's agent migration behaviour and devise an efficient migration strategy and agent redistribution scheme to minimize outbound communications. Though our messaging optimization improved performance, it handles only local messaging. We therefore also work on the optimization of cross node message delivery.

One of the limitations of this study is that the use of a synthetic workload does not show all the relevant issues. We therefore plan to use our findings to improve the scalability of an existing JADE-based MABS application from the transportation and logistics domain [17].

9. References

[1] A. Drogoul, D. Vanbergue and T. Meurisse, "*Multi agent based simulation: Where are the agents?*" Lecture Notes in Computer Science, Multi-Agent-Based Simulation II, 2581 2003, 43-49

[2] The Foundation of Intelligent Physical Agents. http://www.fipa.org, visited June 2008.

[3] B. Page and W. Kreutzer, "*The java simulation handbook*", Shaker Verlag, Aachen Germany: 2005.

[4] The Madkit Project. http://www.madkit.org visited June 2008.

[5] F. Bellifemine, A. Poggi and G. Rimassa. *JADE A White Paper. http://jade.tilab.com/papers/2003/WhitePaperJADE-EXP.pdf,* Visited March 2008.

[6] European Coordination Action for Agent-based Computing. http://www.agentlink.org/ Visited June 2008.

[7] M. Nowostawski et.al. "*Platfroms for agent oriented software engineering*", Proceedings of Seventh Asia Pacific software engineering conference, Singapore, 2000, 480-488.

[8] P.M. Ricordel and Y. Demazeau, "*From analysis to deployment: a multi-agent platform survey*", Lecture Notes in Computer Science, Multi-Agent-Based Simulation, 1972 2000, 93-105

[9] R. Tobias and C. Hofmann, "*Evaluation of free Java-libraries for social-scientific agent based simulation*", Journal of Artificial Societies and Social Simulation, 7(1), http://jasss.soc.surrey.ac.uk/7/1/6.html

[10] D. Mengistu, P. Davidsson and L. Lundberg, "*Middleware Support for Performance Improvement of MABS Applications in the Grid Environment*", Lecture Notes in Computer Science, Multi-Agent-Based Simulation VIII, 5003 2008, 20-35

[11] C.D. Pham "*Comparison of Message Aggregation Strategies for Parallel Simulations on a High Performance Cluster",* Proc. 8th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, USA 2000, 358-365.

[12] X. Zhang, J.L. Freschl, J.M.Schopf. "*A performance study of monitoring and information services for distributed systems*",

Proceedings of The 12th Int'l Symposium on High Performance Distributed Computing. Seattle, USA 2003, 270-281.

[13] X. Wang et. al, "*Measurement and analysis of LDAP performance",* IEEE/ACM Transactions on Networking 16(1), 2008.

[14] K. Jurasovic, G. Jezic and M. Kusek, "*A performance analysis of multi-agent systems*", International Transactions on Systems Science and Applications, 1(4), 2006.

[15] G. Vitaglione, F.Quarta and E. Cortese, "*Scalability and Performance of JADE Message Transport System*", Proc. AAMAS Workshop on AgentCities, Bologna, Italy, 2002

[16] D. Chen et. al. "*Large scale agent based simulation on the Grid",* Science Direct Journal of Future Generation Computer Systems 24(7), 2008, 658-671.

[17] J. Holmgren, et. al. "*An Agent Based Simulator for Production and Transportation of Products*", 11th World Conference on Transport Research. Berkeley, USA. 2007

[18] K. Chmiel et. al. "*Testing the efficiency of JADE agent platform*", Proceeding of International Symposium on Parallel and Distributed Computing, Ireland, 2004, 49-56.

[19] F. Wang, S.J. Turner and L. Wang. "*Agent communication in distributed simulation*", Lecture Notes in Computer Science, Multi-Agent-Based Simulation III, 3415, 2005, 11-24

[20] Theodoropoulos G, et. al. "*Large scale agent-based simulation on the Grid",* Proc. 6th International Symposium on Cluster Computing and the Grid, 2(1), Singapore, 2006.

SIX

Performance Modeling and Optimization of Agent Based Simulations in Distributed Environment

Dawit Mengistu, Paul Davidsson, Lars Lundberg, Peter Tröger
Department of Systems and Software Engineering
School of Engineering, Blekinge Institute of Technology,
372 25 Ronneby, Sweden

E-mail: {dawit.mengistu, paul.davidsson, lars.lundberg, peter.troger}@bth.se

Abstract

Many multi-agent based simulation (MABS) applications are very large and are not suitable for execution on a single computer. However, they often have performance and scalability problems in a distributed computing environment due to their high communication-to-computation ratio. To overcome this problem, existing communication optimization approaches may be employed. However, in some cases, due to the characteristics of MABS applications, the overhead incurred by the optimization operation itself can outweigh the saving. It is therefore important to know the conditions in which an optimization strategy can be useful or becomes a liability. In this paper, we present a performance modeling based approach to evaluate optimization strategies in distributed MABS applications. The approach provides a mechanism for online estimation of runtime and application tuning. We demonstrate that substantial reduction in application runtime and communication traffic can be achieved using the proposed prediction models.

Keywords: *Simulation, MABS, Performance Modeling, Distributed Computing.*

1 Introduction

The amount of computing resources needed to execute a multi agent based simulation (MABS) application designed to study a social system depends on the size of the system to be studied. For certain types of studies, the invariance of the findings drawn from agent based simulations cannot be assured if the results and conclusions depend on the size of the simulation [16]. For this reason, it is often required to undertake a full-scale simulation by modeling the behaviour of all entities of the system and the interactions among them. A MABS of such a scale often cannot be carried out in reasonable time on a single computer and sometimes huge computing resources are required.

Affordable computing resources capable of executing applications of such magnitude are available in the form of parallel and distributed systems such as clusters and the Grid. Optimization of MABS applications in distributed environment is critical because parallelized MABS applications generally have a well known problem of high communication-to-computation ratio [2]. This problem is worse in loosely coupled systems such as clusters and the Grid where data exchange take place over network paths of high communication latency. Unless optimization mechanisms are implemented, the performance loss due to communication bottlenecks outweighs the expected benefits from distributed processing. It would therefore be impossible to achieve application scalability and the anticipated gain in execution speed.

One of the recommended approaches to improve communication performance include deployment of MABS agents in such a way that as much inter-agent communication as possible can take place among agents deployed in the same node. However, this approach requires prior explicit knowledge of agent communication graph, while in reality this graph often changes as the simulation progresses. Grouping and redistribution of agents based on the evolving interaction pattern requires generating the communication graph and migrating agents across nodes on the basis of their group behaviour. Another approach is the use of message aggregation techniques where smaller messages to the same destination can be grouped into a larger one before leaving the origin node. Both approaches cause additional computational overhead. Migrating agents involves studying the self-organization and group formation behaviour in the MABS. Besides, migration is a computationally expensive task in distributed execution environments. Message aggregation entails the use of an additional software component (or a communication

agent) to manage messages on the basis of their destinations, dispatching outgoing messages and delivering incoming messages in a similar manner. Due to differences in interaction behaviour among agents, the resulting intensity of communications and the associated overhead to apply these approaches, it is not possible to obtain the same level of performance yield in all MABS scenarios.

In this paper, we propose a prediction model based approach to make informed decisions on performance improvement alternatives in distributed MABS applications. We show that performance prediction models can be used for application level optimizations of communication overhead in MABS, to improve scalability, to enable dynamic load balancing and an on-the-fly application tuning. We have built performance prediction models for common MABS communication architectures in different execution environments. The rest of this paper is organized as follows: a brief overview of related work is presented, followed by the methodology applied in this work. We then discuss the design of application workloads used in our experimental setup used to build performance prediction models. In the Results section, we present the prediction models and evaluation of the performance optimization strategies. We then discuss the important findings of this research and conclude the paper by identifying directions of future work.

2. Related work

A large body of literature about performance prediction models is available. Several tools exist for application and resource monitoring, and performance modeling of large-scale scientific applications. However, there is not much information available on performance prediction for MABS as it is a relatively recent entry into the distributed computing arena. Existing prediction models cannot be directly applied to MABS as they have the following limitations:

- The granularity of tasks they deal with is much higher than those of MABS;
- They expect well defined inter task communication and execution flow, while MABS execution is not generally deterministic;

Ripeanu et al. propose a performance prediction approach for the Cactus toolkit in the Grid environment [7]. This work gives a useful insight into the mathematical prediction models for large scientific simulations.

Pham compares different message aggregation strategies in parallel simulations [2]. The aggregation concept is similar to ours, but the recommendations on how to pick the most efficient strategy is not directly relevant to MABS applications in general and to our proposed model based approach in particular.

Vitaglione et.al [12] studied scalability issues for the JADE multi agent platform in a distributed environment under different configurations and intensity of inter-agent communication. Their work gives a general insight into the architecture and performance of the message transport systems in distributed multi agent platforms.

A dynamic agent allocation and load balancing strategy for distributed multi agent applications is given in the work by Jang and Agha [21]. A multi agent framework called Adaptive Actor Architecture (AAA) is implemented to monitor agent communication patterns, group behaviour and load conditions to execute agent migration actions to improve performance.

In our earlier work, we proposed an agent migration algorithm based on heuristics to improve the communication performance of MABS in a Grid environment [3]. The approach anticipates performance gains from migration considering that the simulation will thereafter run with a lower communication overhead for a sufficiently long time, eventually offsetting the migration cost. However, these assumptions may not hold for some simulations. Moreover, it is not possible to validate the optimality of the migration heuristics.

Our current work employs performance prediction models to estimate the expected gains from agent migration and evaluates different performance improvement schemes to determine an optimal one under prevailing operating conditions. It extends the modeling approaches in other Grid based scientific applications to include special features of agent based simulations. As we shall show, the models provide better insights into communication optimization in distributed MABS and facilitate dynamic and online agent reallocation decisions.

3. Methodology

The review of performance-related works shows that message aggregation and communication load balancing through migration are the approaches offering ease of implementation and improvement of the communication behaviour of distributed simulations. We undertake a study based on experiments to further increase the performance gains achievable through these approaches.

MABS applications are not usually developed from scratch. They are deployed on either a multi-agent system (MAS) platform, or an agent based modeling and simulation (ABMS) tool. Our choice of platform in this work is the Java agent development framework (JADE) platform, an open source software familiar to the agent community [12]. It offers low level functionalities to enable measurement, analysis and improvement of communication performance, besides its support for application development, deployment and partitioning. Furthermore, it has a well maintained documentation that is regularly updated.

On the application part, we designed a simulation model that runs on JADE. To investigate the communication behaviour of MABS applications in a realistic setting, we ran the experiments in both homogeneous and heterogeneous environments. We designed two representative workloads containing the essential features of typical MABS applications. The communication models are based on the two common agent interaction topologies of existing MABS workloads [1]: peer-to-peer and hierarchical. The application (business) logic of the simulation, i.e., the behaviour of individual simulation agents is modeled in the form of generic computation intensive operations of measurable length.

MABS applications are not suited for centralized performance data collection and modeling due to the overwhelming communication traffic they generate. We introduced some features into the simulation platform to minimize the need to communicate with the node hosting the main platform (central node). One of these features is to keep copies of the white and yellow page directory information at every node to minimize directory requests and handle inter-node messages without involving the central node. The most important function of the central node under normal conditions is then maintaining time step synchronization globally. We applied a novel approach where performance models are iteratively built at individual compute-nodes

and consensus is achieved among all nodes without loading the network significantly.

We performed several experiments to collect sufficient data to build the desired performance prediction models. The models were then evaluated and validated with additional measurements.

4. Workload Design

For the peer-to-peer workload, group formation problem in social networks was considered. For the hierarchical case, we considered a problem in the transportation and logistics domain where the simulated system is modeled as a multi-level organization.

Mathematical model of the workload

We consider a simulation size of N agents deployed on M machines such as nodes of one cluster or the Grid. We assume that MABS are executed as time-driven simulation applications. Thus, if the simulation is run for nts time steps, the total execution time will be

$$T_{sim} = p.\ n_{ts} \qquad (1)$$

where p is the duration of one time step.

Each time step has two phases, a computation phase and a communication phase. In the computation phase, a simulation agent executes program code corresponding to the tasks of the real world entity it simulates. In the communication phase, it may send (and receive) one or more messages to (and from) other agents. To maintain causality, the two phases are synchronized centrally and clearly separated so that no agent falls behind or advance ahead of the rest of the group. It then follows that

$$p = t_{comp} + t_{comm} + t_{sync} \qquad (2)$$

where t_{comp} and t_{comm} are the computation and communication times in one simulation time step respectively, and t_{sync} is the idle time that the agents

spend waiting for the central synchronization signal to advance to the next time step.

If N = the simulation size (number of agents),
M = number of computing nodes
G= the mean length of an agent's task in the computation phase (also called task granularity),

Assuming all nodes host the same number of agents and neglecting the thread context switching overhead (too small compared with G), we have

$$t_{comp} = N.G/M \quad (3)$$

The granularity G depends on the performance of individual compute nodes. The communication time further depends on the number of agent peers deployed on the same node and the latency of the communication network. The number of messages sent over the network expressed as a percentage of the overall inter-agent messages is defined as the outbound communication ratio, r. If the initial agent allocation is carried out in a purely random manner, then

$$r = 1 - (1/M) \quad (4)$$

In simulations where a priori information about the behaviour of the agents is available, MABS applications may use *smart* load generators to apply informed initial allocations so that the actual r is minimized.

The workload will therefore have four parameters, N, M, G, r (where r can be changed during a simulation by agent migration). In the experiments, we varied the values of these parameters for each interaction topology to estimate the runtime p.

5. Experiment Design

We performed two sets of experiments, one on a homogenous cluster and the other in a heterogeneous environment consisting of a cluster and remote nodes connected over the Internet. The cluster is a Linux-based system with

five PIII 1GHZ processor nodes with 1GB RAM. The remote nodes used in the heterogeneous case are a mix of Windows and Linux machines with a similar configuration connected via Internet backbone and a 100Mbps switch.

The simulation is launched as a master-slave application with the cluster's head-node used as the central node. A set of runtime data is collected by executing the workload, with the following range of parameter values for the respective interaction architectures:

- Simulation size N: varied from 500 to 8000 agents, incremented in steps of 500 for each measurement instance.
- Number of compute nodes M: took values 2, 3, 4, 5 machines on a cluster, two more nodes were added for the Grid experiments. Initially, all compute nodes host equal number of agents.
- Computational granularity G: took values 1, 2, 3, 4, 5 ms.
- Outbound communication rate r: varied from 10% to 90% in steps of 10 for each measurement instance.

A set of measurement data was collected taking different combinations of the above parameters. We used the average execution time of one simulation step (p) in our performance metrics. For each input combination, we run the simulation for 20 steps. We discarded the first five measurements (to remove simulation warm up time effect) and computed the average of the remaining to get the corresponding value of p.

5.1. Peer-to-peer

The peer-to-peer performance model was developed from the model of group formation simulation in social networks as follows. It is assumed that the emergent dynamics of the simulated system leads to the formation of k clusters of agent groups where the agents in each group are expected to interact in a peer to peer fashion. The distribution of the members of a group across the compute nodes is uniform.

For a group g_i with n_i member agents $a_{i1}, a_{i2}, \ldots, a_{in}$, let n_{im} be the number of agents located on compute node m. Furthermore, let an agent a_{ij} of this group have interaction with p_j peer agents.

For our experiment, we chose the set of peer agents in such a way that the ratio of the number of peers hosted on the same node to that of p_j is equal to the outbound communication rate r. To achieve this, the simulation was executed for sometime, until a steady clustering behaviour is obtained. The agents then subscribe their group membership through JADE's yellow page directory service. The application analyzes the clustering information by processing the yellow page entries and generates a simulated peer-list for each agent in such a way that a controlled outbound communication rate can be achieved.

During one simulation time, each agent sends out a message to one of the agents in its peer list and waits for a reply to update its state information and knowledge of the environment. There will therefore be *2N* messages transmitted through the system out of which *r*100 %* will be outbound.

5.2. Hierarchical

To build the performance model, a case from an ongoing research in the transportation and logistics domain was studied. The problem we considered employs a MABS approach for transportation policy analysis using a simulator tool called TAPAS (Transportation And Production Agent-based Simulator) [17]. The agent interaction follows a four-level hierarchical organization with a Transport Chain Coordinator (TCC) agent at the top, a Product Buyer (PB) and Transport Buyer (TB) agents below it. The PB agent has Production Planner (PP) agents under it, while the TB agent has Transport Planner (TP) agent below. At the lowest level, we have agents modeling factories/manufacturing sites, transporters and customers.

In this application, the higher level agents were deployed on the compute nodes based on the input parameters of the experiment in such a way as to allow control of the outbound communication. However, it is not always possible to determine the interaction pattern a-priori as it may be an emergent phenomena itself.

The inter-agent interaction is initiated by messages sent from customer agents to higher levels requesting for products and transportation for the products. The recipients in their turn send messages to the agents one level above them in the organization. Reply messages are then sent back from the

top level agents to the bottom level in the reverse order. Higher level agents combine requests and replies from lower levels and pass to the upper ones.

The major difference between peer-to-peer and hierarchical interactions from performance point of view, is that in the peer-to-peer case the interaction is evenly distributed among all agents, while in the hierarchical case, they are concentrated towards agents at the middle layers. As a result, we expect different mathematical models to predict the performance of the two cases.

6. Results

In each run, the mean execution time p of one simulation step is obtained from execution time measurements. The analysis of measured data shows that the duration of the computation phase of p, t_{comp}, increases linearly with the number of agents deployed on a node. The synchronization time, t_{sync}, is dependent on the total number of nodes used in a simulation, N. There is also an additional overhead added by the MAS platform even if the application were executed on a single node. In our setup, this time was about 45μs per agent deployed on one compute node. The duration of the communication phase, t_{comm}, depends on r and the latency of the network. Our measurements show that it increases non-linearly with N.

We used Matlab to analyze the data and evaluated different polynomial mathematical models for the prediction model to get an estimate of p in terms of N. The closest estimate was obtained with the following quadratic model:

$$p = \alpha_2 N^2 + \alpha_1 N + \alpha_0 \qquad (5)$$

where the coefficients α_i $(i = 0,1,2,3)$ are approximated from historical data of the workload execution. From our measurements, for $M = 5$, our prediction model for the peer-to-peer topology becomes:

$$\begin{aligned} \alpha_2 &= 0.3959\, r - 0.01224G - 0.0109r^2 + 0.0289\, G^2 \\ \alpha_1 &= 0.2613\, r - 0.\ 1242G - 0.22616r^2 - 0.4352G^2 \\ \alpha_0 &= 0.1619r + 0.2157G - 0.05141r^2 + 0.0843G^2 \end{aligned} \qquad (6)$$

The prediction model was validated with a series of measurements and comparing the predictions with measured data as can be seen in figure 1 and 2. It is observed that the prediction is accurate enough, with the worst error not exceeding 11%.

We repeated the above experiment in a heterogeneous environment with packet round trip time of 28msec. Initial results of data analysis show that the corresponding performance model is not as accurate as the cluster environment, but is still useful.

Figure 1. Validation of model on a 5-node homogeneous cluster (r = 0.7, G=3)

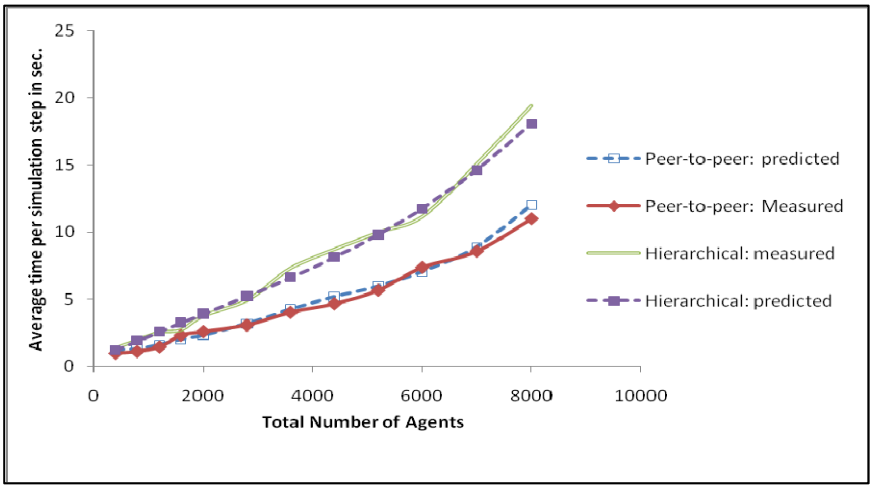
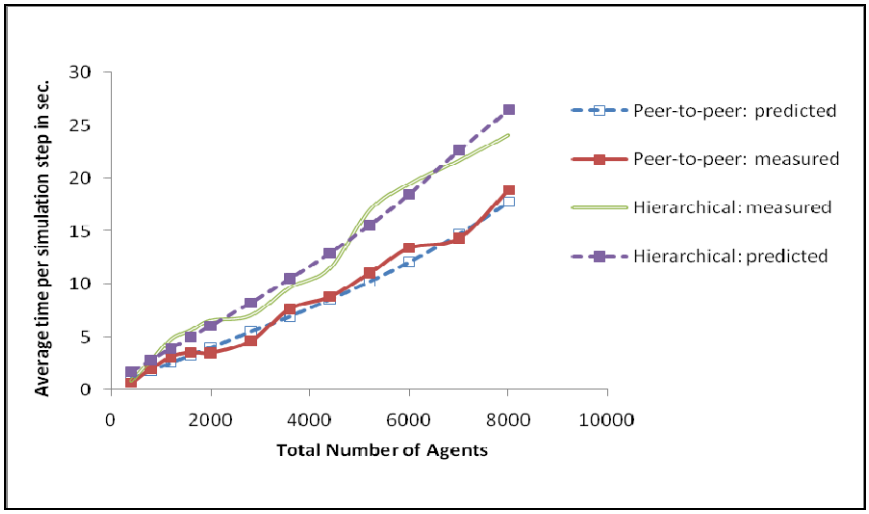


Figure 2. Validation of model in a heterogeneous environment (r = 0.7, G=3)



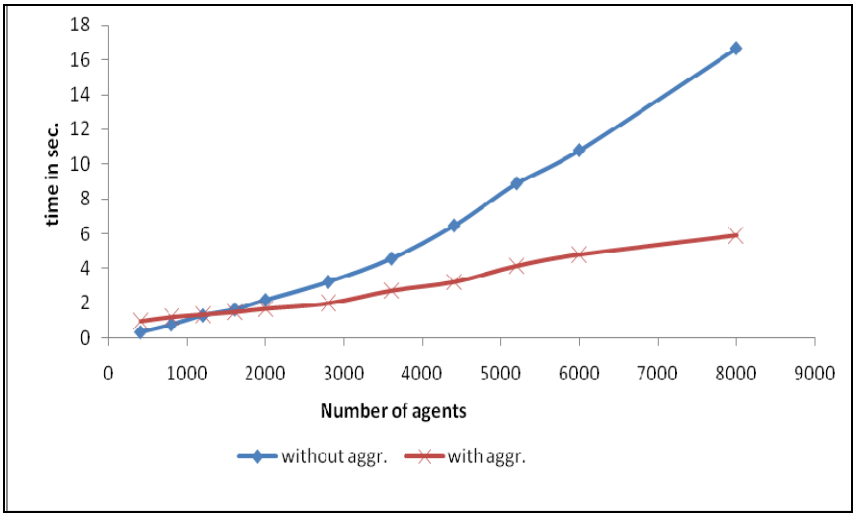
The above figures show that larger communication overhead occurs in hierarchically organized models. Message queues will be formed at higher levels agents as they receive messages from several lower level agents simultaneously and need more time to interact with each of them.

6.1. Optimization using message aggregation

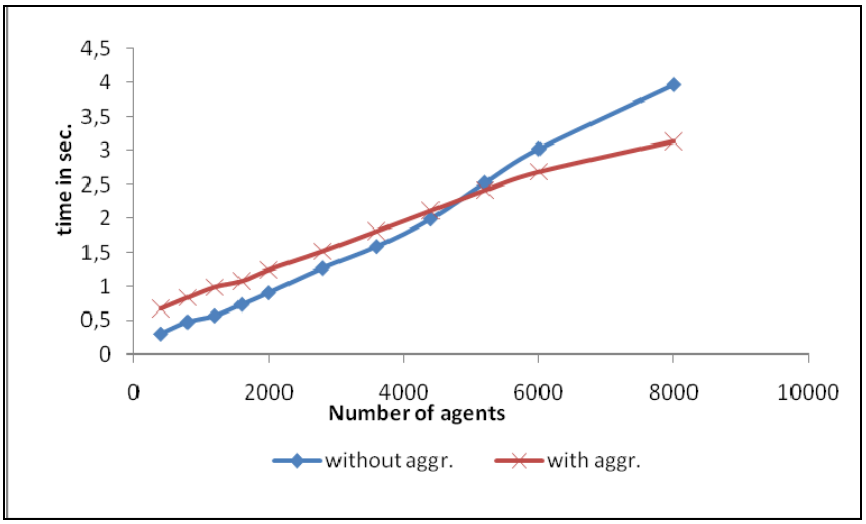
In message aggregation, individual messages are not sent to agents at other nodes on the instant of creation. Instead, they are temporarily stored locally and sorted by their destination node, and dispatched in batches to reduce contention over network resources. The delivery of messages is also handled similarly. This minimizes the communication overhead significantly, particularly in applications where a large number of messages with small size are produced [2]. The down side of this approach is that it introduces overhead associated with storage, sorting and delivery of messages. Its viability depends on the simulation size and the intensity of communication among agents deployed on different nodes as can be seen in figure 3.

Another performance model similar to equation (5) is built with the message aggregation component included. For a simulation having a certain *r, G and N,* the execution time estimates are compared for the two models to determine if message aggregation is appropriate for the scenario at hand. In large-scale simulations with a high value of *r,* the execution time will be reduced substantially. In figure 3a, for example, the runtime of one simulation step is reduced by about 80% with a simulation size of 8000 agents for a peer-to-peer interaction architecture.

Figure 3. Evaluating the performance of message aggregation on a homogeneous cluster



a. r=0.7, G=1



b. r=0.1, G=1

As can be observed from figures 3a and 3b, the crossover point for the two curves depends on all workload parameters. The performance prediction models are used to determine whether the message aggregation yields a good optimization for a workload under a given operating conditions.

6.2. Optimization using agent migration

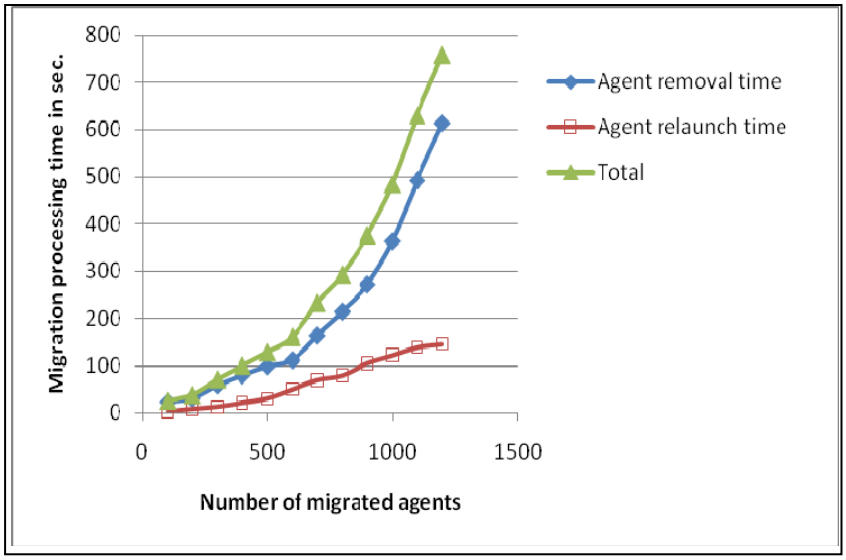
Comparing the runtimes in figures 3a and 3b, one can see that reduction of *r* improves performance significantly even if message aggregation may not be used. Assuming the inter-agent interaction pattern to be approximately static random, i.e., if the simulation agents show a relatively steady communication pattern, the corresponding entity-interaction graph will consist of fairly distinct clusters of agents. It will then be possible to co-host agents on the basis of the clustering information so that *r* takes on a smaller value. Most of inter-agent messaging will thus be memory copy operations. The implementation of this approach involves:

a. Obtaining the communication graph and generating agent clusters by analyzing the inter-agent message traffic and planning reallocation schemes;
b. Identifying the agents to be migrated and saving their recent states;
c. Terminating and removing the to-be-migrated agents at their current node;
d. Transferring the states to the new destination;
e. Launching the migrated agents at the new locations with their old state information and updating the agent directory services to reflect the outcome.

While these operations are performed, the simulation has to be suspended, and if any of the operations is not successful, the original states are restored. The JADE platform offers agent migration interfaces, but the performance is good enough only when the number of agents to be migrated is small [4]. Migration process as detailed above, undoubtedly entails significant overhead, and its viability should be weighed in before applying it. Unless the gain in the remaining simulation time offsets this overhead, the whole exercise will be useless.

A breakdown of the major components of migration overhead is shown in figure 4. The contributions of *a, b* and *d* are not shown in the figure as they are mostly in-memory operations and involve only a one-time data transfer over the network. The migration model is independent of any of the system parameters except the simulation size, or the number of agents to be migrated.

Figure 4, breakdown of migration overhead compared to the number of agents moved



A prediction model of migration overhead for our experimental workload is:

$$T_{mig} = 0.0007N^2 - 0.2146N + 58.916 \quad (7)$$

Let p_{pre} and p_{post} respectively denote the estimated execution times of a simulation step before and after migration.

If r_{pre} and r_{post} respectively represent the corresponding outbound communication rates, and the simulation is scheduled to run for k_s steps, the total remaining simulation time T_{sim} with and without migration can be determined using equation (1) as:

$$T_{sim}^{mig} = k_s \,.\, p_{post} \quad (8a)$$

$$T_{sim}^{nomig} = k_s . p_{pre} \quad (8b)$$

The predictor would recommend migration if

$$T_{sim}^{mig} < T_{sim}^{nomig}, \quad (9)$$

For equation (8) to hold, it is required that the migration overhead is offset with the saving in the total simulation time. It then follows that migration is advantageous if the number of remaining simulation steps satisfies the following condition:

$$k_s > T_{mig} / (p_{pre} - p_{post}) \quad (10)$$

The predictor judges whether migration is a worthy action in the given situation by substituting the new value of the would-be achievable *r* in the performance model equations (5, 6) and estimating the remaining simulation time. If the migration effort outweighs the performance gain, a new set of agent clusters with less migration will be iteratively computed and the model is re-evaluated until a feasible migration scheme is reached. However, since the iteration is based on heuristics, it may not always yield an optimal migration scheme.

The migration model validation experiments show that the predictions are reasonable if the number of migrated agents at a time is not too high. For example, when we tried to migrate 300 agents from each node, the operation could not successfully complete due to network congestion and exhaustion of system resources. One way to overcome this problem over a sufficiently large remaining simulation time is to carry out the migration in two or more steps.

6. Discussion

One of the challenges of prediction modeling with MABS applications is the lack of centralized control in application execution. Simulation agents are autonomous and their communication pattern is not predictable in short

simulation runs. One should therefore collect large performance data to build a reasonable model.

The prediction models built in this experiment cannot be generalized for all MABS applications. However, a close look at equation (5) gives some useful clues about estimation of resource requirements for a simulation. In finely grained simulations, because the execution time depends mainly on *r* rather than on G, execution on powerful compute resources would not achieve a meaningful saving in execution time. Simulation modellers can therefore take this into consideration during resource reservation to save on CPU hour costs whose rates depend on processor speed.

The prediction models can also be used for static load balancing by facilitating an informed input data generation in some situations to avoid arbitrary deployment of agents on compute nodes. We observed that in hierarchical organizations, deploying higher level agents on separate nodes reduces traffic significantly.

In peer-to-peer simulations, group formation and sections of interaction graphs are locally generated at the worker nodes. This removes the need for a centralized management of agent migrations and introduces self-organization features.

7. Conclusions

This work demonstrates how model based optimization can make MABS a feasible workload in distributed environment. The major findings reported here have several uses. MABS users and application developers can make informed design decisions on the architecture of their applications to improve scalability. Estimates of resource requirements, automated generation of simulation load can be assisted by the models to produce efficient initial agent allocation and deployment schemes for distributed MABS applications.

The selection of an appropriate optimization technique is not trivial. The size of the simulation application, the number and length of simulation runs, the number of compute nodes, the intensity of communication, the amount of computational task agents execute, etc., affect the choice of an appropriate strategy. We have shown how a performance prediction model can be used to make a reasonable decision to optimize the execution of MABS applications.

The optimization strategies themselves incur computational overheads that may at times not justify using any of them. The message aggregation technique adds its overhead in small amounts throughout the simulation execution, while the overhead of the migration is a large one that occurs only during the time of migration. Comparing both approaches, the magnitude of migration overhead is generally much higher, particularly if the reallocation moves around a large number of agents.

In our next step, we will port the experiment to cloud computing resources, to build a more comprehensive prediction model to be used in a production environment. We also plan to validate our findings further with real workloads.

One of the observations in migration based optimization is that, finding the optimal migration scheme from the communication graph is an NP-hard problem. We see that this is also an interesting research challenge to tackle in our future work.

8. References

[1] Davidsson, P., et al.: Applications of multi-agent based simulation, Lecture Notes in Computer Science, Multi-Agent-Based Simulation VII, Vol. 4442, 2007, 20-35

[2] Pham, C.D.: Comparison of Message Aggregation Strategies for Parallel Simulations on a High Performance Cluster, Proc. 8th Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, USA 2000, 358-365.

[3] Mengistu, D, Davidsson, P. and Lundberg, L.: Middleware Support for Performance Improvement of MABS Applications in the Grid Environment, Lecture Notes in Computer Science, Multi-Agent-Based Simulation VIII, Vol. 5003, 2008, 20-35

[4] Bellifemine, F., Poggi, A. and Rimassa, G.: JADE A White Paper. http://jade.tilab.com/-papers/2003/WhitePaperJADEEXP.pdf, Visited March 2008.

[5] Wang, F., Turner, S.J. and Wang, L.: Agent communication in distributed simulation, Lecture Notes in Computer Science, Multi-Agent-Based Simulation III, 3415, 2005, 11-24

[6] S.A. Jarvis, D.P. Spooner, H.N.L.C. Keung, G.R. Nudd: Performance prediction and its use in parallel and distributed computing systems, Proc. 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA 2003

[7] Ripeanu, M., Iamnitchi, A., Foster, I.: Cactus application: performance prediction in Grid environments, Lecture Notes in Computer Science 2150, 2001, 807-816

[8] Zhang, X., Freschl, J.L., Schopf, J.M.: A performance study of monitoring and information services for distributed systems, Proc. The 12th Int'l Symposium on High Performance Distributed Computing. Seattle, USA 2003, 270-281.

[9] Holmgren, J., et. al: An Agent Based Simulator for Production and Transportation of Products, 11th World Conference on Transport Research. Berkeley, USA. 2007

[10] Chmiel, K. et. al: Testing the efficiency of JADE agent platform, Proc. Int'l Symposium on Parallel and Distributed Computing, Ireland, 2004, 49-56.

[11] Jurasovic, K., Jezic, G. and Kusek, M.: A performance analysis of multi-agent systems, International Transactions on Systems Science and Applications, 1(4), 2006.

[12] Vitaglione, G., Quarta, F. and Cortese, E.: Scalability and Performance of JADE Message Transport System, Proc. AAMAS Workshop, Bologna, Italy, 2002

[13] Page, B. and Kreutzer, W.: The Java simulation handbook, Shaker Verlag, Aachen Germany: 2005.

[14] Theodoropoulos G, et al: Large scale agent-based simulation on the Grid, Proc. 6th International Symposium on Cluster Computing and the Grid, 2(1), Singapore, 2006.

[15] Mengistu, D. and Tröger, P.: Performance Optimization for Multi-agent Based Simulation in Grid Environments, 8th IEEE International Symposium on Cluster Computing and the Grid. Lyon, France 2008.

[16] Cioffi-Revilla, C.: Invariance and universality in social agent-based simulations. Proc. National Academy of Science USA, 99 (2002) Suppl. 3: 7314-6

[17] Davidsson, P., Holmgren, J., Persson, J. and Ramstedt L.: Multi Agent Based Simulation of Transport Chains, Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2008.

[18] C Sansores, C. and Pavon, J.: A framework for agent based social simulation. The Second European Workshop on Multi-Agent Systems, Barcelona, Spain, 2004.

[19] The Net Logo Home Page. http://ccl.northwestern.edu/netlogo/ Visited January 2009.

[20] REPAST Agent Simulation Toolkit. http://repast.sourceforge.net/ Visited January 2009.

[21] Calzarossa, M., Massari L. and Tessera, D.: Workload characterization issues and methodologies. Computer Science, Performance Evaluation, Vol.1769, 2000, 459-482.

[22] Jang, M. and Agha, G.: Adaptive agent allocation for massively multi-agent applications, Lecture Notes in Artificial Intelligence, MMAS, Vol. 3446, 2005, 25-39.

[23] The Foundation of Intelligent Physical Agents. http://www.fipa.org, visited June 2008.

SEVEN

An Algorithm for Optimistic Distributed Simulations

Dawit Mengistu
School of Engineering, Blekinge Institute of Technology
Soft Center, 372 25 Ronneby, Sweden
dawit.mengistu@bth.se

Martin v. Löwis
Hasso Plattner Institute, Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam
martin.vonloewis@hpi.uni-potsdam.de

Abstract

Distributed simulation has traditionally suffered from synchronization constraints that reduce the speedup significantly below the theoretical maximum. One solution approach are optimistic synchronization algorithms, allowing some nodes to advance in model time. In this paper, we present a novel algorithm that does not share the typical rollback costs of optimistic simulation, at the expense of restrictions on the design of simulation models. We have implemented that algorithm in an agent-based simulation framework, and evaluate its performance gains through measurements on actual simulation models.

KEYWORDS
Optimistic Simulation, Discrete Event Simulation, Agents

1. Introduction

One of the requirements of a distributed simulation is preserving the causal behavior of the simulated events, in order to create a virtual model of some real world system it is intended to mimic. A simulation should accurately capture the dynamics of the simulated physical process without violating the temporal sequence of the events. The model of the system to be simulated is divided into its logical components, deployed and executed on different machines. When an event in one of the components is simulated, other events may be triggered in the remaining components. To maintain the temporal order of the execution of these events, we need a proper synchronization mechanism between the logical components.

Conservative and optimistic synchronizations are employed as the two major time management approaches in event-driven distributed simulations [1] [2] [20]. In both approaches, the simulation components are considered as logical processes that communicate with each other through time stamped messages. A message initiates the simulation of events at the receiving process. The conservative synchronization approach makes sure that a process always simulates events with smallest time stamp value so that causal ordering is not violated. Optimistic approaches, in particular time-warp based algorithms, follow a method which does not always guarantee causality [3]. When a violation occurs, the process rolls back to resume the simulation from the most recent correct state.

The methods used in the conservative approach result in degradation of execution performance and lower processor utilization because the execution follows a nearly sequential behavior. Allowing violation as in the optimistic approach on the other hand is sometimes too costly. Flaws in temporal order of events executed at one process may trigger invalid execution sequences at other processes too, the correction of which results in cascaded rollbacks [3].

In this paper, we present a tailored optimistic time management approach to improve the performance of distributed simulation, specifically of multi agent based simulation (MABS) applications. MABS is a discrete event simulation technique suitable for studying the dynamics and emergent behavior in poorly understood complex systems lacking generalized mathematical models. MABS applications are often implemented as time driven simulations with simulation time advancing in fixed steps (usually unit) [4].

This paper is organized as follows: an overview of MABS and its usual time synchronization approach is presented in the next section. We then discuss the related work for the topic, discuss our approach and present an experimental work to validate our proposal. The paper is concluded by highlighting the major findings of this work and a discussion of directions for future work.

2. Overview of Synchronization in MABS

In MABS, the simulation artifacts are software agents characterized by autonomy, decentralization of data and absence of a global system control mechanism. Agents have important capabilities such as proactive execution of tasks and roles, and the ability to cooperatively solve complex problems that are beyond an individual's capability. They have also the ability to interact with each other to update their states and knowledge of their environment (the term 'belief' is commonly used in agent lingo). Agent communication may take place in two ways: either through explicit message passing between agents, or, through a shared blackboard-based information update.

Evolutionary behavior and emergent phenomena in the simulation model are outcomes of the interaction among agents. These features make the MABS technology suitable for study of complex systems.

Building agent based applications from scratch is time consuming as they often lack consistency and uniformity. MABS are therefore usually developed and executed using dedicated agent based simulation tools or generic multi agent platforms. The tools and platforms provide basic agent templates, low level system APIs, and support for directory services, message transport, standard agent interaction protocols, and time management.

MABS are identified by the presence of some kind of logical organization of the agents according to their functions. This organization is in most cases role oriented and defines the interaction links that can exist between agents. Agents are often organized in hierarchical or peer-to-peer fashion following the structure of the system they model.

Owing to the way in which simulation time advance is modeled, MABS can be implemented as either time-driven or event-driven. Event-driven

simulations require a central coordinator to keep track of the events to be executed in causal order. As the notion of a central coordinator contradicts with the principal tenets of MABS (absence of centralized control), the time-driven approach is the logical implementation model [4]. It is often convenient to realize time-driven models with the following constraints to ensure causality [5]:

1. The execution of role tasks (henceforth referred to as computation) and updating of belief by an agent are interleaved throughout the simulation. An agent updates its belief state by communicating with other agents and exchanging state information. During one simulation time, the computation phase is performed first and the communication follows next. In the communication phase, agents send out and/or receive state information as messages.
2. The advance of simulation time is controlled conservatively to avoid violation of causality. During the communication phase, an agent has to stay in *wait* state until it receives the anticipated state update information. This denotes the state-of-the-art in agent frameworks.

Although violation of causality is avoided by preserving the temporal order of event execution, the application does not scale well particularly for large simulations.

In our previous implementation of distributed MABS, each machine participating in the simulation hosts several agents. To enforce synchronization, a global barrier is used. In large-scale simulations which involve thousands of agents, the synchronization is carried out in two steps to reduce communication overhead. Agents running on one machine perform their execution up to a local barrier only. When all agents synchronize at their respective local barriers, a manager agent on each machine synchronizes with fellow managers at other machines through multicast messages, triggering the advancing of simulation time. Therefore one global time is shared by all participants of the simulation. Figure 1 illustrates the execution of one simulation time step in a distributed MABS.

The dotted vertical lines in Figure 1 show the time instances at which local synchronization occurs at the processing nodes while the shaded vertical rectangle represents the global synchronization time. The positive aspects of this model are its straight forward implementation and memory efficiency (avoiding to save multiple states). However, this synchronization model brings a significant performance penalty. Some processing nodes stay idle in

their local barrier, waiting for the slower nodes to reach the global barrier and advance the simulation time.

The proportion of this idle time to the overall simulation time is affected by other factors such as the intensity of inter-agent interaction, network traffic and the computation rate of different nodes. Because MABS already have high communication-to-computation ratio, this particular synchronization style compounds perfor-mance degradation.

Execution at processor nodes does not take place at the same pace due to the following reasons:

1. It is usually difficult to achieve perfect load balancing and thus uniform execution rate across all processor nodes.
2. Communication latency and network traffic affect message delivery and hence increase idling time.
3. Even if ideal conditions can be met to solve the above two issues the most serious problem is, balancing the communication load. A processor node stays idle for a long time while the agents wait for state update. This factor is dependent on the dynamics of the simulated system.

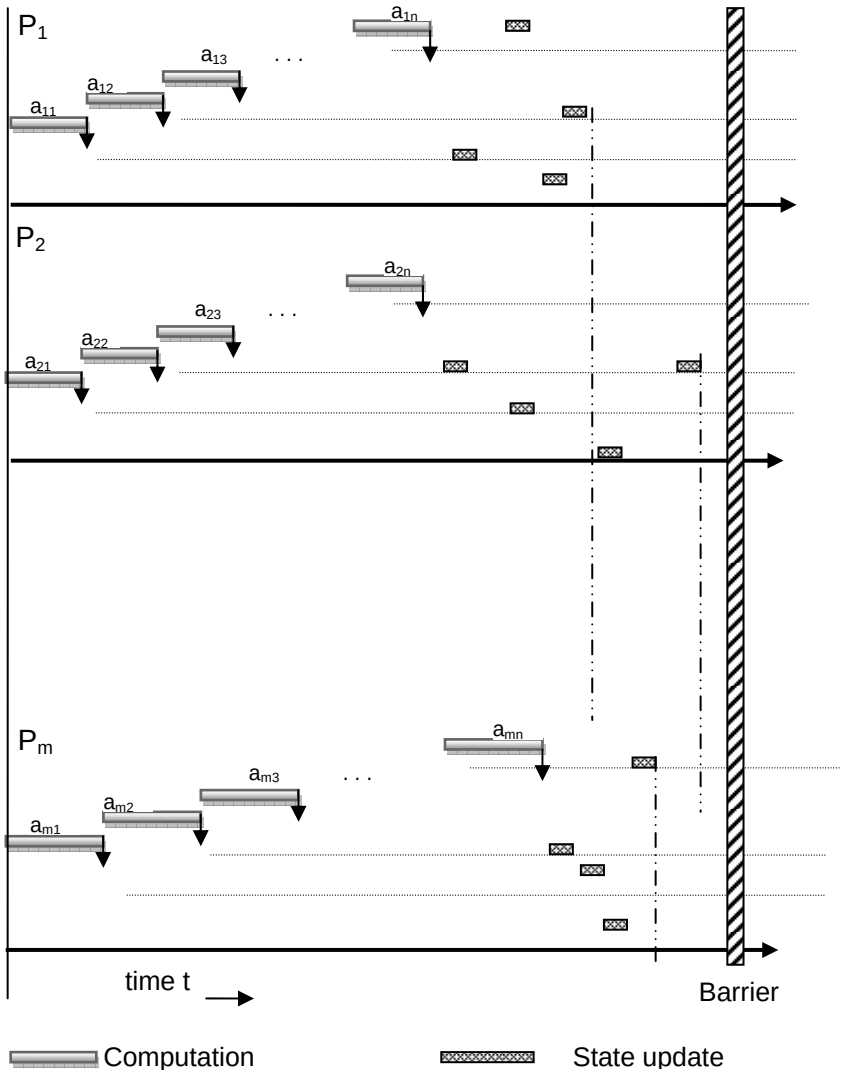


Figure 1. Execution flow for one simulation time in conservative synchronization:

$P_1, P_2, ..., P_m$ are processing nodes (could be on the same physical machine or distributed)

a_{ij} is the j^{th} agent deployed on node i.

One more source of longer idle times is the thread scheduling policy of the operating system at the processor nodes. Even if the scheduler attempts to allocate CPU time to agent threads fairly, the allocation sequence may be in conflict with the desired execution sequence of the simulation. The problem is more visible in finely grained simulations and is reflected in the form of longer waiting time in the communication phase.

The alternative approach of optimistic synchronization, such as with *Time Warp,* that allows violation of causality and correction of wrong states through rollback does not suit common MABS implementations [3]. Complex systems are usually characterized by the intensity of interaction among their entities, a feature that translates into a dense agent communication graph on the simulation application. In such cases, causality violations occur frequently and are likely to trigger cascaded rollbacks. Figures 2 and 3 show the communication graphs of the two common MABS interaction architectures: peer-to-peer and hierarchical. The nodes represent agents and the edges are drawn between interacting nodes.

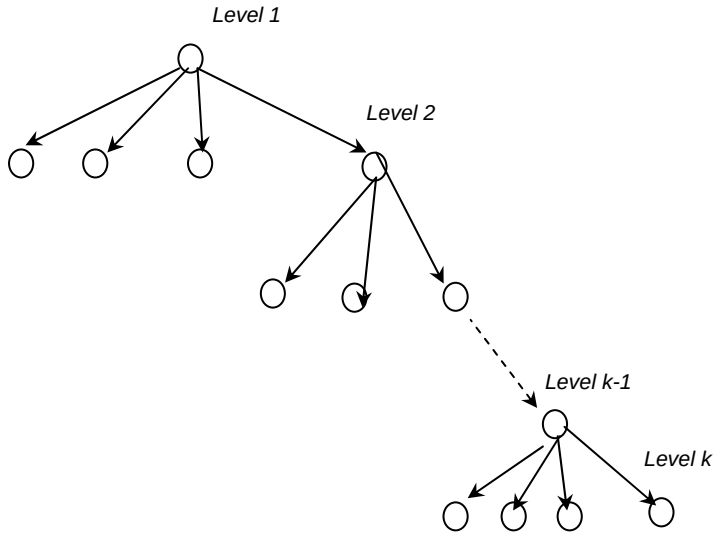


Figure 2. A k-level hierarchical communication

Let

N = the simulation size (number of agents)

n_i = the number of agents immediately below an agent at level i in a hierarchical architecture

If a causality violation forces an agent at level i to roll back, the number of cascaded rollbacks *(R)* in the worst case will be

$$R = \prod_{j=i}^{k} n_j$$

If the edges in the graph are undirected, we will have the worst case:

$$R = N$$

It can also be seen on the peer-to-peer graph that depending on the intensity of interaction, rollback can be an expensive operation. Causality violation by agents in groups A and B is likely to affect the simulation more severely than those in C or D.

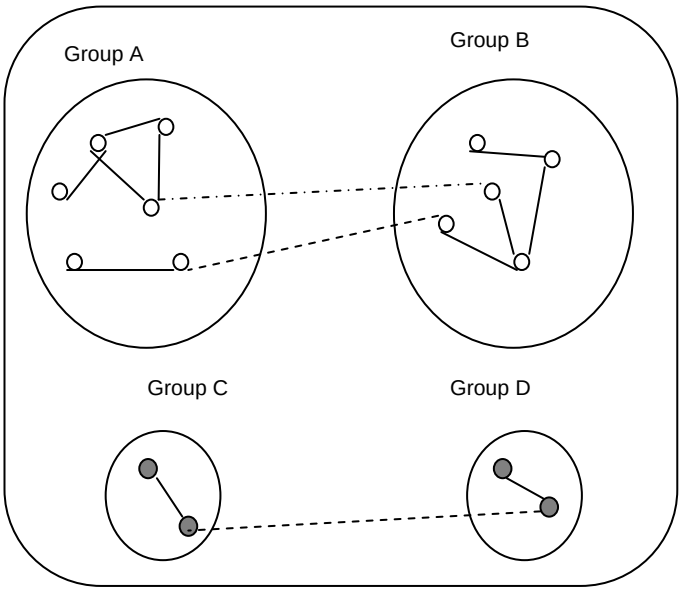


Figure 3. Peer-to-peer interaction

3. Proposed Approach

Our approach exploits the idle time at the nodes by keeping agents executing without waiting for the barrier. In contrast to other solutions, it also avoids violation of causality and hence rolling back. We introduce the following changes in the time management part of MABS applications to achieve these objectives:

1. We remove both local and global synchronization barriers and allow ready-to-run agents to advance local simulation time directly.
2. Agents creating state updates by receiving messages from other agents are forced to stay in a wait state until they receive complete information and are not allowed to advance the simulation time otherwise.
3. State updates are *only* received through reply messages for requests sent out by an agent to its companions.

The first principle helps to reduce the idle time wasted on waiting for a synchronization signal while the next two ensure that causality is not violated and useful computation time is not wasted on rollbacks. This is possible through the restriction of agent state updates by message reception only. In this approach, there is an implicit requirement that an agent expects messages from known sources only. Many MABS applications have the features needed to satisfy the requirements of these principles.

First, standard multi agent platforms have directory services such as yellow pages [5]. It is possible to define the interaction graph using these services to enable agents identify their peers, super- or sub-ordinates at any instant throughout the simulation. Second, agents are able to initiate communication proactively and send requests for state updates, and decide what course of action to take by themselves.

In classical optimistic event-driven simulation, the logical processes do not know in advance the source and thus the time of arrival of the next event. Furthermore, if the event arrives in ‘past’ time, all events processed after that time should be cancelled and rerun. This distributed rollback activity by 'anti-messages' is the major cost factor in the classical optimistic approach.

In contrast, our approach does not allow the arrival of past messages that impact the course of the next action. It remains true that all agents at the processor nodes may not advance the simulation time at the same pace since they execute their tasks asynchronously. Each agent should thus maintain its

own clock to control the simulation time and sequence of events and messages. Accordingly, all inter agent messages should be time stamped so that the receiver can deal with them in temporal order.

If an agent sends a request to a slower peer, it cannot receive a reply until the slower peer advances to the same temporal value as the sender, were it is able to create an appropriate answer. Similarly, when the slower agent receives a 'future' message from its faster peers, the message should be saved until such a time that the receiver is able to reply.

Another scenario is the reception by a faster agent of a 'past' request, although this request is sent by a slow sender in current time. In order to send a causal reply to late messages, the requested party needs to save its past states in memory.

Memory Constraints:

For long running simulations, it is not practical to save all states and future messages indefinitely as the application would run out of space. We consider the following scenario to estimate the amount of space needed.

For a simulation size of N agents executed on a system with B bytes of available memory,

Let

d = the number of time steps the slowest agent lags behind from the fastest agents,

b_s = the maximum number of bytes needed to save state information.

For saving state, the worst case is that one agents lags behind, and all other agents are ahead of GVT byt the same distance d. The worst case memory space requirement S_s will thus be:

$$S_s = d * (N-1) * b_s \qquad (1)$$

For communication, the worst case is that all agents but the slowest one have outstanding read requests to other agents, eventually cascading to wait on that slowest agent. If b_m is the maximum number of bytes needed to store a future message, in the worst case, we need

$$S_m = (N - 1) * b_m \quad (2)$$

bytes of additional memory for the same scenario.

The total memory needed is the sum of *(1)* and *(2)* above. However, for long running simulations the time gap between the extreme agents could be large so that

$$Sm << Ss \quad (3)$$

We will then have the maximum allowable lag by the slowest agent to be:

$$d = B / ((N-1) * bs) \quad (4)$$

All agents must comply with the memory constraint and keep no more states than the above. For this purpose, we define a global virtual time (GVT) to be the local simulation time of the slowest agent. To obtain GVT, we first establish a local virtual time (LVT) for each processor node. Because it is expensive to update the GVT at every instant agents advance their local times, a good heuristic is for an agent to update the LVT every d_a steps where

$$d_a = d/m. \quad m = 2, 3, 4, \ldots$$

Each node should receive the LVT of all other nodes to determine the GVT, which is the smallest LVT of all nodes. A node must send multicast messages to update the GVT when its LVT has advanced by d_a steps from the last LVT it communicated to other nodes. An agent whose simulation time progressed by d steps from GVT should not perform its task until GVT advances.

4. Examples

Our algorithm can be used to optimize the performance of different classes of distributed MABS applications. Typical scenarios include the following:

1. Electronic market simulations: in a typical market simulation, the roles of the main entities such as buyers, sellers, etc., are modeled with agents [17]. These agents register themselves in the platform directory services and advertise their roles at the start of execution. A buyer agent comes to know its potential list of suppliers (neighborhood, product type, etc) by searching in the yellow pages. Buyers initiate request for quotation or price negotiations. Our algorithm is applicable in markets in which prices vary with time (season). It, however, does not guarantee rollback free execution in situations where demand based price fluctuations occur often. In such cases, one needs to reduce the value of d in equation (4) to a reasonable empirical value corresponding to the instants of price updates.

2. Spatially explicit simulations: In simulation of phenomena such as crowd, epidemic, search-and-rescue operations, etc., agents assume physical space in an environment, and change their locations and states as the simulation progresses. Agents can read and change the states of the environment by executing a sense-think-act routine and communicating through a blackboard messaging system[18]. Deployment of such applications in a distributed system requires partitioning the simulation task by dividing the environment into regions (usually of equal size) mapped to processing units. Conservative synchronization is inefficient because state updates at the partition boundaries require exchange of data between compute nodes. With our optimistic approach, however, most of the agents are usually situated far from the boundaries, and can therefore proceed with their tasks for a long time. It is therefore possible to run the application efficiently by avoiding idle waiting for boundary state updates.

In certain MABS scenarios, the execution time slows down close to the conservative approach, and substantial optimization cannot be achieved with the proposed algorithm. This is particularly the case in hierarchically organized supply chain simulations involving a series of communications between agents at different levels. Some attempts are made to use a hybrid time management approach in logistics MABS, to apply domain specific knowledge in combination with time warp [19]. The limitation of this approach is, however, that the number of rollbacks may not be tolerable for large simulations involving thousands of agents.

5. Experiment and Results

The trade-off between the different synchronization approaches is of interest to evaluate their effectiveness and limitations. We conducted an experimental work to evaluate the viability of our approach in comparison with existing approaches. Because MABS applications are modeled and implemented in diverse ways, we setup a MABS workload which contains the most common features of MABS discussed in Section 3. We used this workload in our earlier performance optimization studies of distributed MABS applications [5][6]. Our workload model contains the following features:

1. The agent platform Yellow Page service registers the necessary attributes of agents such as their type, spatial location (if any), etc., which help identification of fellow agents.
2. An agent communicates with one or more of its fellows obtained from the yellow page directory to receive state updates.
3. The state update request and reply may take a form of conversation where multiple exchanges can take place before the requester gets all the information it needed. We apply agent communication protocols specified by the Federation of Physical Agents (FIPA) to direct the flow of the conversations [7]. FIPA specifications contain a set of message primitives used in inter-agent conversations.

Agents achieve their objectives through interaction, negotiation and cooperation. Their communications usually take place based on protocols defined by standardization bodies.

The Java Agent Development Framework (JADE) multi agent platform was used in our experiments. JADE provides the desired functionalities and ease of parallelizing the simulation application. We studied the performance of a peer-to-peer architecture for agent interaction. The variable parameters of our experiment are:

- the number of neighbors an agent has, and
- the amount of computation an agent executes to perform its role.

The experiments were conducted on an a shared memory machine with 8 cores (4X2). The shared memory machines have better message handling

capabilities but do not scale well. The cluster machines have poor communication performance but are easily scalable. It would therefore be interesting to evaluate the performance of the proposed approach in both environments.

To balance the simulation load, the number of agents deployed on all processing nodes was the same. In the first set of experiments, each agent has up to 3 friends and performs a computation of 1msec in one time step. The simulation is run for 100 time steps.

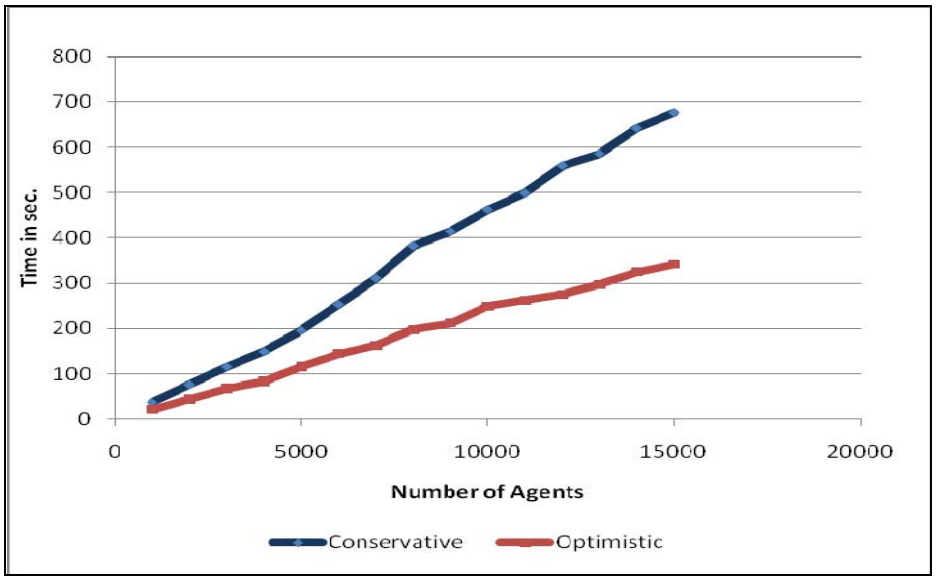


Figure 4. Comparison of execution of optimistic and conservative algorithms.

In Figure 4, we compare our original implementation with our new optimistic approach, on the multi-core system. We run the same simulation multiple times, with the system size being the only parameter. We configure the simulation for a fixed period of model time which the model needs to execute, and measure the wall clock execution time. As can be seen from the figure, both algorithms have a linear increase of time depending on the problem size (which is plausible given that each agents needs to perform a fixed amount of work per time step). In both the conservative and the

optimistic approach, the amount of computation is the same, but the wall-clock time is lower by roughly a factor of two in the optimistic case. Consequentially, CPU utilization is higher in the optimistic case.

6. Related Work

Jefferson [9] pioneered the notion of optimistic simulation, and the notion of virtual time [1] for discrete event simulation. Various contributions have been made to improving the algorithms for state saving in Time Warp ([10], [11], [12]). All these approaches are orthogonal to the one presented here – in our algorithm, we still need an efficient state saving algorithm, and many of the proposed ideas apply to our algorithm as well.

A number of approaches have been proposed to make the rollback of Time Warp more efficient, e.g. by means of reverse computation ([13], [14]). In comparison, our solution avoids rollbacks altogether.

Several proposals have been made to incorporate synchronous behavior into optimistic simulation, similar to our case where a read operation for a future state of a remote agent blocks synchronously until that state is available. Jha and Bagrodia [15] proposed a framework where each node can choose whether to operate in optimistic or conservative mode. In our approach, no such explicit selection is necessary; instead, whether the agent can operate conservatively or optimistically depends on its relative progress in the complete simulation. Xu and Zhang [16] propose that a process can execute in three modes: synchronous, conservative, and optimistic. However, they primarily use this decision to avoid state saving when it is known that rollbacks cannot occur, and still let rollbacks happen when in optimistic mode.

7. Conclusions

We have presented a tailored optimistic time management approach for distributed agent-based simulation. This approach avoids the need for

rollbacks which has traditionally been associated with optimistic simulations, while still allowing agents to advance far ahead of global virtual time, and still requiring the state saving necessary to go back in time if a message from the past arrives. We also integrate conservative elements in this approach, by having agents block when they request state from agents that have not advanced in virtual time enough.

This novel algorithm is able to provide significant speedups as opposed to traditional conservative models. We have created a formal model of performance aspects of the approach, and were able to demonstrate them in actual simulations.

The algorithm is restricted to cases where messages from the past cannot affect the behavior of the agent that receives the message, i.e. are *read* message from the viewpoint of the receiving agent. We believe that the approach is viable for a significant number of simulation models, and have proposed criteria under which our approach applies.

Further studies need to focus on reformulating existing simulations in our framework; our implementation needs to integrate more efficient approaches for state saving, building here on results achieved by other researchers.

8. References

[1] D. R Jefferson, Virtual time. *ACM Transactions on Programming Languages and Systems 7*(3) 1985. 404–425.

[2] J. Misra, Distributed Discrete Event Simulation. *ACM Transactions on Computing Surveys, 18*(11), 1986. 39-65.

[3] A. Ferscha, Parallel and Distributed Simulation of Discrete Event Systems. *Parallel and Distributed Computing Handbook. (ed) A.Y. Zomaya*, (McGraw-Hill New York 1996 ISBN 0-07-073020-2).

[4] P. Davidsson, Multi Agent Based Simulation: Beyond Social Simulation. *Workshop on Multi Agent Based Simulation (MABS)* 2000, LNAI, Springer, 97-107.

[5] D. Mengistu., P. Davidsson, and L. Lundberg, Middleware Support for Performance Improvement of Multi Agent Based Simulations. *Multi*

Agent based Simulation VIII 2008, LNAI Springer Berlin/Heidelberg, 20-35.

[6] D. Mengistu, and P. Tröger, Performance Optimization for Multi-agent Based Simulation in Grid Environments. *International Symposium on Cluster Computing and the Grid*, France 2008, 560-565.

[7] FIPA Agent Management Specification, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. http://www.fipa.org/specs/fipa00023/SC00023J.html visited June 2009.

[8] D. R. Jefferson. Virtual time II: Storage management in distributed simulation. *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing* (Aug. 1990), ACM, New York, NY, 75–89.

[9] D. Jefferson, and H. Sowizral. Fast Concurrent Simulation using the Time Warp Mechanism, Part I: Local Control, *Rand Note-1906AF*, The Rand Corporation, Santa Monica, California, 1982.

[10] J. Fleischmann, and P. Wilsey. Comparative analysis of periodic state saving techniques in time warp simulators, *Workshop on Parallel and Distributed Simulation*, Lake Placid, New York, 1995.

[11] Q. Liu., and G. Wainer. Lightweight Time Warp- A Novel Protocol for Parallel Optimistic Simulation of Large-Scale DEVS and Cell-DEVS, *Models, Distributed Simulation and Real-Time Application*, 2008.

[12] D. Bauer., and E. Page. An Approach for Incorporating Rollback through Perfectly Reversible Computation in a Stream Simulator, *21st International Workshop on Principles of Advanced and Distributed Simulation*, 2007.

[13] A. Naborsky., and R. Fujimoto. Using Reversible Computation Techniques in a Parallel Optimistic Simulation of a Multi-Processor Computing System, *21st International Workshop on Principles of Advanced and Distributed Simulation*, 2007.

[14] S. B. Yoginath, and K. S. Perumalla, Parallel Vehicular Traffic Simulation using Reverse Computation-based Optimistic Execution, *Principles of Advanced and Distributed Simulation*, 2008.

[15] V. Jha and R. L. Bargrodia, A Unified Framework for Conservative and Optimistic Distributed Simulation, *Workshop on Parallel and Distributed Computing*, 1994.

[16] J. Xu, and J. Zhang, Efficiently unifying parallel simulation techniques, *ACM Southeast Regional Conference*, Melbourne, Florida, 2006

[17] P. Mariano et.al., Simulation of a trading multi-agent system, *IEEE International Conference on Systems*, Man, and Cybernetics, USA, 2001

[18] M. Lees, B. Logan, R. Minson, T. Oguara, G. K. Theodoropoulos. *Modeling Environments for Distributed Simulation, 1st International Workshop on Environments for Multi-Agent Systems (E4MAS)*, in conjunction with the *3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AA- MAS04) 2004.

[19] D. Pawlaszczyk, I. Timm, A Hybrid Time Management Approach to Agent Based Simulation. *Proceedings of the 29th Annual German Conference on Artificial Intelligence*, LNCS 4314, Springer, Berlin 2006.

[20] K. M. Chandy, J. Misra, Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering, 5*(5), 1979, 440-452.

EIGHT

Approaches to Distributed Multi-Agent Based Simulation

School of Computing, Blekinge Institute of Technology
Soft Center, Box 520 37225 Ronneby, Sweden
Dawit Mengistu, Paul Davidsson, Lars Lundberg
{dawit.mengistu, paul.davidsson, lars.lundberg}@bth.se

Abstract

Multi-agent based simulation (MABS) has emerged as a promising alternative to traditional equation based macro simulations techniques in the study of poorly understood phenomena. However, the pace of its adoption in simulation of large systems is not satisfactory. One of the reasons for the slow progress is that most of the publicly available MABS tools are single threaded desktop applications that are not parallelizable. Even the few parallel MABS that exist are not scalable as their underlying platforms were not designed for large-scale simulations. This paper presents three different approaches for implementation of MABS applications in distributed execution environment. We propose parallelization strategies for single threaded MABS tools and cite use cases that benefit from it. We identify some sources of performance bottlenecks on existing parallel tools and their solutions. We also propose a new approach for distributed MABS application development based on the web services technology. Our experimental studies show that scalable distributed application can be built on top of single threaded MABS tools, while the execution time of existing parallel tools can be reduced by up to 60% by employing performance tuning techniques.

KEY WORDS: *MABS, Distributed Simulation, Multi-Agent Systems, Agent Based Simulation.*

1. Introduction

The emergence of distributed computing paradigms such as cluster and Grid has stimulated the development of new scientific applications. Newer distributed computing infrastructures and programming models have helped the realization of platforms for large-scale applications. An application area potentially benefiting from distributed execution is large-scale simulation, particularly multi-agent based simulation (MABS).

MABS is used for studying and explaining the dynamics of complex systems composed of entities that interact with each other. In this paper, we will focus on the development and deployment of MABS applications for efficient execution in distributed environments. A major step to achieve this objective is to understand the architectural features and the execution behavior of MABS applications. We thus explore how existing and future MABS implementations can be parallelized and distributed. In principle, there are three different approaches to achieve this:

1. Parallelizing and distributing existing MABS tools
2. Using distributed multi-agent system platforms
3. Using general distributed computing techniques

There are a number of special-purpose tools available for developing MABS applications. These tools have several important features such as agent model templates, prototypes for generic simulations, graphical interfaces for simulation visualization. They also provide mathematical modules needed for analysis purposes. However, most of these tools are implemented as single-threaded sequential applications.

Multi-agent system (MAS) platforms are general-purpose packages for developing multi-agent applications. Common MAS platforms support multi-threading and distributed execution and, are thus potentially suitable for distributed MABS applications.

Distributed computing resources are available to users as co-located clusters of homogeneous machines, large-scale grids, public and private clouds. With the ubiquity and maturity of Internet technologies, these resources are becoming increasingly accessible to simulation users. However, because the resources run on diverse hardware and software platforms, they were not effectively exploited in the MABS domain. In order to address the interoperability problems, we propose to use the web services technology for

MABS application development, deployment and execution in distributed environment.

In this paper, the characteristics of the above approaches are evaluated through experiments. The contributions of this work are three-fold: first, it examines available distributed programming models and services to parallelize MABS applications. Second, it addresses the sources of performance bottlenecks that need to be overcome in order to effectively harness the potential of distributed computing resources. Third, it identifies MABS use cases and scenarios that can be handled with a particular architectural model and implementation strategy, and hence which type of deployment and execution model suit best for each case. In future sections, we shall present these contributions and the experimental work on which our results are based.

The rest of this paper is organized as follows: a quick review of multi agent based simulation is presented next, followed by a discussion on the methodology employed in this work. The experimental studies we have undertaken to investigate the three approaches proposed earlier are presented in Sections 4, 5 and 6 in their order. Related work summarizing publications on MABS application surveys and distributed applications performance is presented in Section 7. We conclude the paper by highlighting the salient points revealed by our research and suggesting areas of further investigation.

2. MABS in Brief

The capabilities of MABS as a simulation approach are evident in experimental studies of complex-adaptive systems that are dynamic and involve communication between interacting parts [3]. The experimental results can be explained and used for prediction purposes. Some researchers describe agent-based modeling as a new way of doing science that has developed from the concepts and techniques of complexity theory [4]. The models may start with simple rules of learning and assumptions but will evolve to levels of higher complexity and reveal emergent behaviours.

A MABS application represents a conceptual model of a real world system. The model consists of the system entities, the set of rules defining their behavior, objectives, and the rules of interactions among entities (at

different conceptual levels). The behavior of the system entities are captured at a micro level in terms of elementary rules and operations. MABS is evolving to a promising technology that helps understand emergent phenomena in several areas of research such as social networks, traffic management, logistics and supply chain management, environment protection, etc. [6, 7].

MABS application development is therefore a multi disciplinary effort that requires knowledge in the research domain of the real world system. The conceptual model is converted to computer programs using high level programming languages such as Java and C++. Agent based simulation kits and agent programming platforms are commonly used to deploy these programs in the execution environment, although in few cases, complete programs are developed from the scratch. Several runs of the simulation with different data sets and configurations are conducted to observe phenomena of interest, to explain the dynamics of the system, to validate a theory, or even build macro models of a poorly understood system.

MABS applications are often desired to study systems involving a large number of heterogeneous entities having complex individual behavior and interaction properties. However, due to lack of adequate computing resources and efficient implementation strategies, users are often obliged to conduct very limited simulations with a small number of agents and a lot of simplifying assumptions. Although in some circumstances partial simulations could suffice to understand the real world phenomena, there exist several systems that cannot be adequately described by limited simulations [9]. It is not always possible to infer a comprehensive conclusion from the outcomes of the partial simulations.

Recent developments in computing have created an opportunity to make advances in large-scale MABS. Distributed high performance computing resources have become a common place to address the demands of the scientific community for solving problems that were once thought to be intractable. Accordingly, the aim of this work is to investigate the possibility of developing large scale MABS applications that execute effectively in a distributed environment, either by porting the existing MABS tools or building new ones.

The fundamental feature of any MABS is capturing emergent behavior that results from the interaction among the entities it simulates [1]. If the entities are deployed on different computers, modeling this interaction may generate a

heavy data exchange among the computers participating in the simulation. Although a large-scale MABS application apparently demands a huge amount of computational resources, our earlier research shows that their resource utilization rate is typically low due to their inherent high communication overhead [2].

3. Methodology

As explained earlier, the principal objective of this work is attainment of efficient execution of distributed MABS. In order to meet this objective, we carried out the following tasks.

1.The state-of-the-art in distributed systems and agent based simulations.

We studied the various aspects of the distributed system paradigm such as programming models, application deployment scenarios and architecture of the execution environment. The main purpose of this study is to determine feasible combinations of distributed programming models, deployment strategies and infrastructure suitable for MABS applications.

The need for performance prediction models for large-scale applications that are executed on distributed resources is well known. Accurate characterization of the execution environment makes it possible to assess the benefits and limitations of distributed systems and to determine whether they can support different application workloads. On the other hand, the huge performance gap between processing power and communication in distributed systems necessitates the use of mathematical models to describe parallel and distributed applications to identify potential bottlenecks, predict their resource demands and estimate execution times [15].

Most of the publications on MABS applications present their works as proof-of-concept to demonstrate the feasibility of MABS as a decision support tool. One can find limited information in the form of implementation issues such as programming and deployment environment. We reviewed different technology surveys and published works [14, 15, 16] on MABS applications. As the surveys have different foci, we had to adopt the findings on the first two to design the application workloads for our experiments and to make a choice of simulation development tools.

2. Experimental analysis of selected MABS application models in a distributed environment.

As mentioned earlier, three implementation models of distributed MABS applications were selected. In the first model, the application is developed using MABS tools. These tools are specifically developed for MABS and thus, users need not have programming skills to use them. In this work, we did not consider proprietary and commercial products because they do not provide full access to low level interfaces needed for this study. We therefore searched for open source MABS tools that are available on the Internet for download. However, literature survey shows that most of the open source tools that are available are usable in desktop environment only [15, 17]. In order to identify a tool that we can adopt to a distributed environment, we conducted test runs on three of the tools. Evaluation of our tests and further review of the documentations showed that, with some work-around and modifications, the NetLogo tool [27] can be used to support distributed MABS execution.

In our second MABS model, the application is deployed and executed on top of a distributed multi-agent system (MAS) platform. The MAS platforms are not originally developed to support MABS applications. One of the features they lack is an explicit mechanism to control simulation causality. The user is required to write proper synchronization code for this purpose. However, they have the necessary agent functionalities and interfaces that are needed to build simulation applications [18]. The presence of such features saves some program development time and helps simulators focus on other essential MABS aspects. Several open source MAS tools are available and some of them have been in use by the MABS community [14, 19]. In this paper, we evaluated relevant platform attributes such support for distributed execution, user friendly API's, documentation clarity and ability of the programming environment to support scalable MABS. Accordingly, we selected the Java Agent Development Framework (JADE) as a simulation platform for this work.

The third model is our own proposed web services implementation of MABS in a distributed environment. The choice of this technology follows the adoption of web services standards in the specification of the Grid services stack. The Open Grid Services Architecture (OGSA) document [21] adopts a service-oriented Grid, and describes the provisioning of required Grid capabilities as services. According to the OGSA specifications, the main components of the Grid such as job execution, resource management, data

management, and security can be implemented as web services. The latest version of Globus, the defacto Grid middleware, is realized as layers of different web services based functionalities [22]. It is also envisaged that the user applications executed on the Grid shall be a collection of web services. Considering the ensuing shift in Grid application development and deployment, it is beneficial to lay the foundation for Grid enabled MABS in accordance with the well established web services guidelines and concepts.

3. Evaluation of MABS implementation approaches.

One factor differentiating MABS from other scientific applications is that MABS applications often have a highly unpredictable execution behavior. Evolutionary phenomena emerging from the interaction among simulated entities usually results in an entirely different set of behavior from the initial settings [46]. Correspondingly, the courses of actions followed by the agents during the execution of the simulation code can drastically vary from agent to agent depending on how the emergent behavior unfolds. It is difficult to build an analytical model for estimating execution time or resource requirements of unpredictable applications such as MABS. We therefore propose to build performance model from observations of application execution under different conditions.

We designed different MABS application workloads for each approach explained earlier. We set up a series of experiments to investigate their execution behavior and characterize their performance in appropriate ways.

4. Parallelization of MABS Tools

As explained earlier, the open-source MABS tools we found fall in the category of desktop applications and are not suited for parallel and distributed execution [17]. The simulation agents run in a single thread and therefore it is not possible to partition the task in a meaningful way. Some tools discuss recent approaches to introduce multi-threading to parallelize the execution of multiple behaviors of a single agent [25]. This alone, however, does not help to distribute the task over several machines, since dispatching the fine tasks of a single agent to several resources results in more problems.

We studied the information on the web sites of some of the tools discussed or cited often in Agent Based Social Simulation (ABSS) survey papers and frequently used by MABS developers. Mason [25], Repast [26] and Net-logo [27] simulation tools were evaluated in this work. In our selection, we considered platform independence and potential multi-threading capabilities of the programming language supported by each tool. We discovered, however, that all tools are single-threaded and cannot be parallelized readily because they were not developed with distributed processing environment in mind. To investigate the effect of this, we ran a simulation from the samples provided with the Netlogo package on a single CPU and a parallel machine to study its processor utilization characteristics. The simulation task was submitted from a remote machine. Figure 1 shows that the tool utilizes not more than 12.5% of the available compute power on an SMP machine having four CPUs with 2 core per CPU.

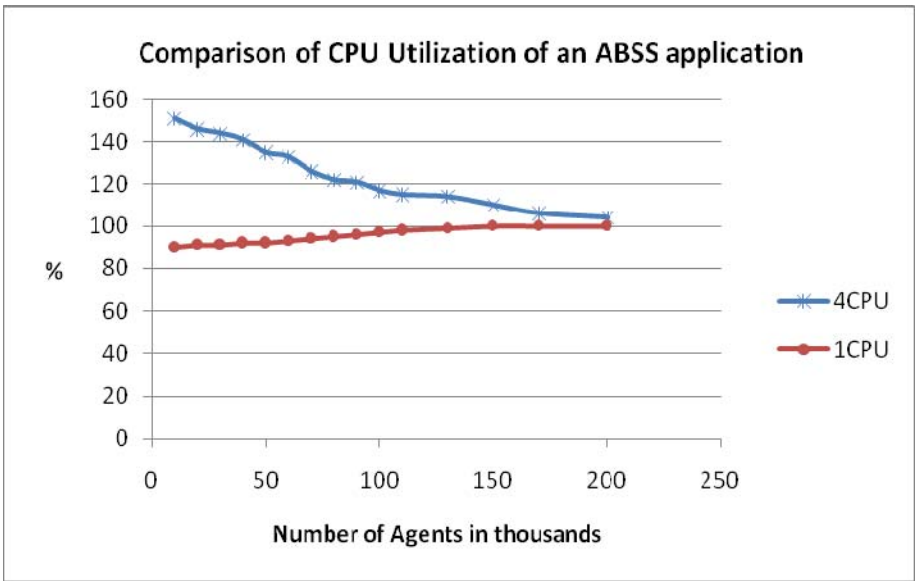


Figure 1. Processor utilization of NetLogo execution

In the figure, the higher utilization for smaller simulation sizes is attributed to network and disk I/O activities.

There are use cases where it is possible to harness the power of parallel and distributed resources and run faster simulations. To identify candidate

MABS applications that can be parallelized, we referred to a survey of published works in MABS conferences [14]. Typical use cases include:

- Simulation based optimizations which can be executed as parameter sweep applications on several resources concurrently.
- Simulations that involve a large number of situated agents with spatial attributes.

It is also possible to creatively redesign other types of simulations to be executed on distributed resources.

Sample simulation examples shipped with the NetLogo tool were customized and used to investigate the general performance characteristics of MABS tools. We used the APIs provided by NetLogo for writing simulations that do not involve GUI support (headless executions as it is referred to in NetLogo). We ran the same test simulation in different environments:

- Individual cluster nodes (single CPU)
- Remote symmetric multiprocessing machines (SMP) with 4 CPUs
- Public cloud

All machines run Java 1.5 and different flavours of Linux and Solaris operating systems. On the cloud nodes, we used 64-bit and 32-bit CentOS Linux versions. The hardware specifications of the machines are as follows:

Resource	CPU			Memory	
	Type	qty	Speed (GHz)	Main (GB)	Cache (MB)
Cluster node	PIII	1	1	1	0,25
SMP-1	Xeon	4X2	2.4	8	4
SMP-2	Sparc	4X2	1.2	16	1
Cloud 64-bit	Xeon	1X2	2.4	2	6
Cloud 32-bit	Xeon	1X2	2.4	2	6

Table 1. Specification of resources used in the experiment

A summary of the execution of a sample simulation on these resources is shown in Figure 2.

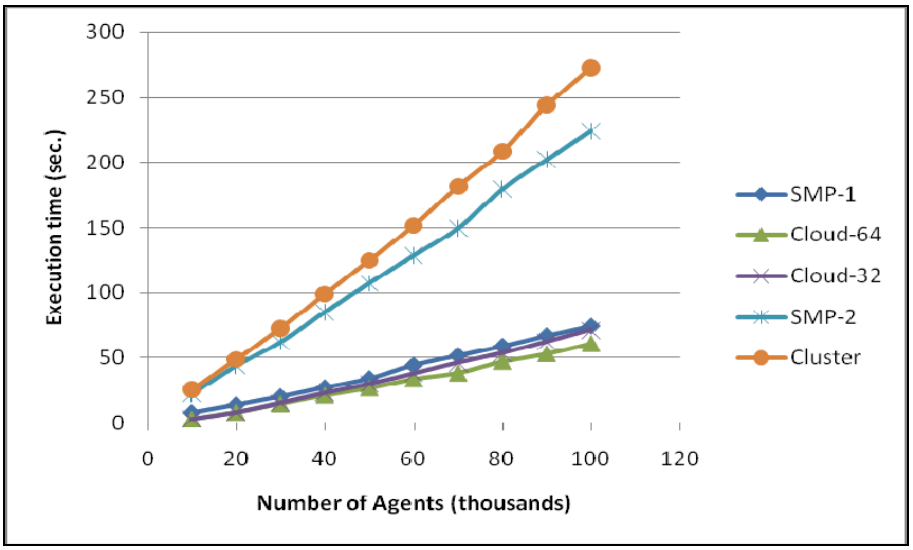


Figure 2 Comparison of different CPUs and execution environments

The execution time increases exponentially on the cluster nodes for very large simulation sizes due to overheads associated with memory swapping as can be seen in Figure 3.

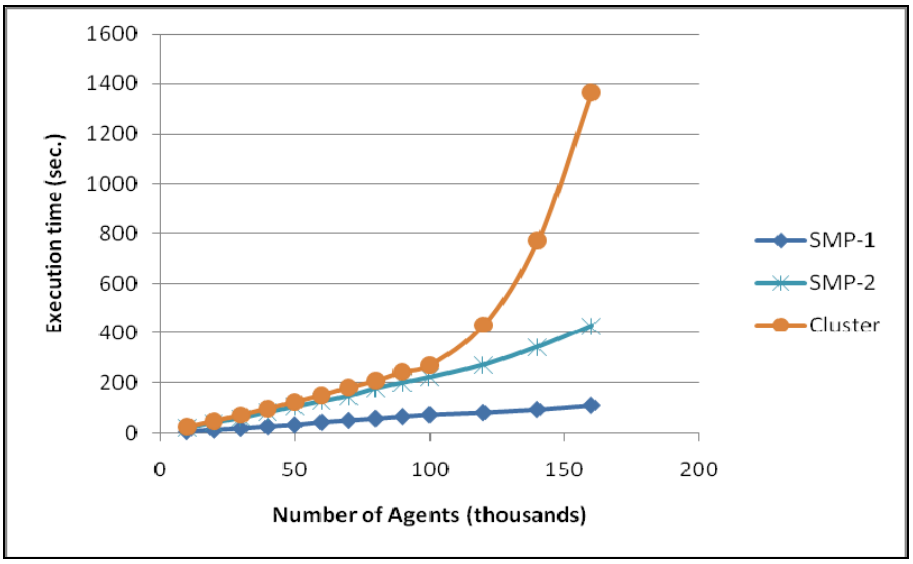


Figure 3 Effect of memory capacity on execution time.

The above figures show that the execution time depends mainly on the CPU speed for a simulation that does not exhaust the main memory and start swapping its data to virtual memory. This is an important finding as it shows that the simulation time can be estimated with a reasonable accuracy by evaluating the application on a user's desktop. We applied this finding to estimate execution times of distributed ABSS applications in a heterogeneous environment as discussed below.

Although the execution environment is not a Grid proper, we implemented or applied the following important features that make the Grid what it is as a distributed execution environment.

1. The notion of a virtual organization by establishing trustee relationships in the form of user and host certificates to facilitate job submission and online load balancing
2. The use of automated scripts to facilitate remote submission and execution of jobs:
3. The enablement of performance monitoring and dynamic load balancing concepts, though the Grid Monitoring Architecture (GMA) specifications were not strictly adhered to.

4.1. Parameter sweep ABSS applications

One area of MABS that can benefit from distributed execution is optimization simulation. In many domains such as transportation and logistics, MABS can be used as a means to find solutions to optimization problems and to provide decision support. The simulation user has to often work with a vast space of feasible solutions. This entails evaluation of the different combinations through exhaustive search or applying some kind of heuristics. The MABS application is executed several times, with different sets of input data combinations each time. The best course of action can be determined after several runs of the simulation by evaluating the results and comparing them with the desired outcome.

The Grid can offer an infrastructure to support parameter sweep applications [30]. In parameter sweep, the application is structured as a set of multiple runs and each run uses a distinct set of parameters as input. To investigate the possibility of using this feature in MABS, we performed experiments in a Grid setting. We ran experiments on two sample simulation

use cases from the NetLogo example library. The first example deals with study of epidemic spreading situations, while the second one was about traffic congestion simulation.

The input parameter space is mapped to a hash table by using an appropriate hashing function. One approach is, to exploit spatial hashing, a technique in which objects in a 2D or 3D domain space are projected into a 1D hash table [44]. Accordingly, the hash function can be modified to map a k-dimensional input space to a collision free hash table. The resulting hash table is then proportionally divided according to the capacities of the computer resources.

Several approaches to dynamic load balancing and adaptive scheduling of such applications are presented in [29, 30]. We recognize the challenges of understanding the behavior of the candidate application and proposing a specific performance improvement strategy. A solution approach for the problem at hand involves:

- Determining the parameter space and the partitioning scheme. Each computer resource gets an amount of work proportional to its processing power. In the case of multi-host resources, the partitions are recursively divided further and the dispatching of workload to individual compute hosts (such as nodes of a cluster) can be automated.
- Wrapping the application for Grid enablement. The simulation can be executed several times with a set of inputs in the parameter space, without the intervention of the user. GUI interactions were disabled and the application was run in the headless execution mode explained earlier.
- Generating the hash table for the parameter space by an appropriate function to get a set of ordered values. If the simulation has k parameters with each parameter taking k_i values, the number of simulation runs is the product $\prod k_i$. We shall pass only two hash values (starting and ending) to each computer resource. The application decodes the task range assigned to it and determines the input parameter set for each simulation run. The task decomposition can thus take place during run time and only the start and end hash values are needed. Furthermore, dynamic adjustment of tasks is possible by changing one or both of these two values. Thus, there is no need to migrate data or task between resources for load balancing purposes.

We consider the following assumptions to enable distributed execution of our application:

1. The user has a preliminary estimate of execution time and resource requirements based on sample runs of the simulation on a desktop machine.
2. Before deploying the application, the user host obtains information about available resources, computing power and current load conditions, from Grid monitoring services or through manual queries.
3. Based on the information gathered about resources, the parameter space is partitioned before the application is deployed. The size of partitions depends on the relative aggregate processing capacity of resources. Multi-host resources further divide their own input parameter space according to the individual node's capacity.
4. The submission host is able to collect performance monitoring data from resources at regular intervals. Multi-host resources aggregate their performance data before sending it to the user.
5. Network latency during data transfer between the submission host and the resources is negligible (is small, compared with the execution time of one simulation run).

Procedure:

1. *Get the parameter space and generate a hash table that maps it into a set of ordered numbers.*
2. *To partition the parameter space, get information on aggregate compute power of each resource and divide the hash table proportional to the task share of each resource.*
3. *Collect monitoring data to estimate an accurate execution time at each resource. Then, predict new execution time and revise the ranges of the hash table accordingly.*

To verify the validity of our approach, we modified an epidemic scenario simulation for large population sizes. Our parameter space is made up of four types of input needed for initialization. The parameter space has 1000 points (*5X5X5X8*) which are mapped in a Hash table. The simulation application is executed on the first two of the resources shown in Table 1. The cluster has 5 nodes and the SMP machine with 2.4GHz processor is used. We first established a preliminary performance prediction model from the measurements shown in Figure 2 based on CPU speed. The initial task-to-resource mapping was thus such that the Hash table rows are partitioned 62.5% and 37.5% for the cluster and the SMP respectively.

The performance model is updated at pre-specified intervals by collecting execution data from the computer resources. The cluster head node aggregates the local performance information and sends it to the user host together with execution output files from each cluster node. The user host then updates the prediction model according to the recent performance data and also revises the load distribution. Each resource is then given new Hash table coordinates delimiting the start and end rows of the parameter space assigned to it. This process continues until the parameter space is exhausted or all scenarios are evaluated.

Results

We repeated the experiment for different balancing intervals and observed the benefit of the adaptive scheme. Figure 4 summarizes the collected information from the experiment.

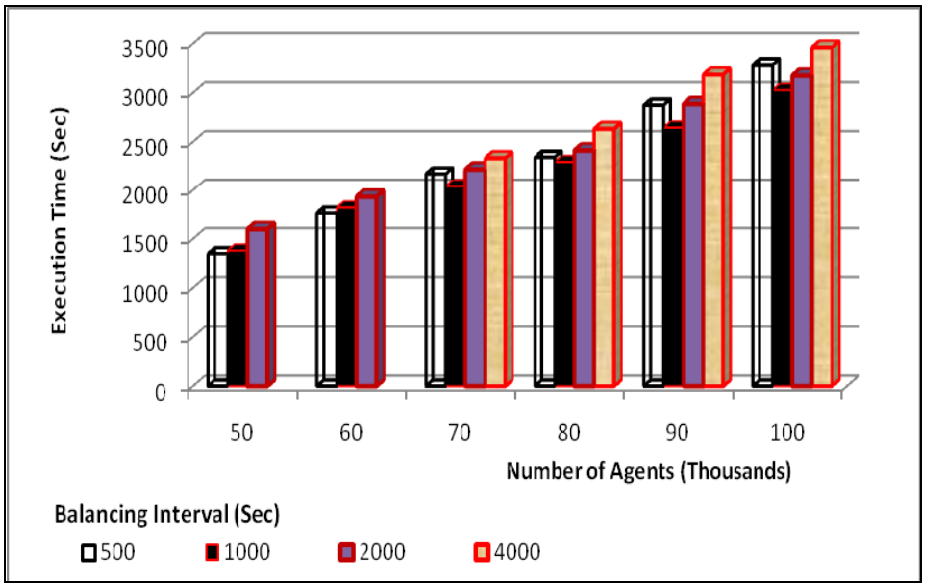


Fig. 4 Execution of a parameter sweep simulation

A balancing interval of 4000 seconds was used to mean no rebalancing since the application execution time is shorter. Figure 4 does not clearly show

whether small or large load balancing intervals is a better choice. We would therefore compare the execution times for different balancing intervals against the no-balancing case as shown in Figure 5. We calculated the percentage of reduction in execution time for the balancing intervals of 500sec, 1000 sec, and 2000 sec. As expected, for smaller simulation sizes, large balancing intervals do not make a significant improvement. Similarly, for large simulation sizes, small balancing intervals contribute extra overhead. In Figure 5, negative values represent a balancing which yielded poorer performance, resulting in longer execution time than the naïve estimate (no balancing).

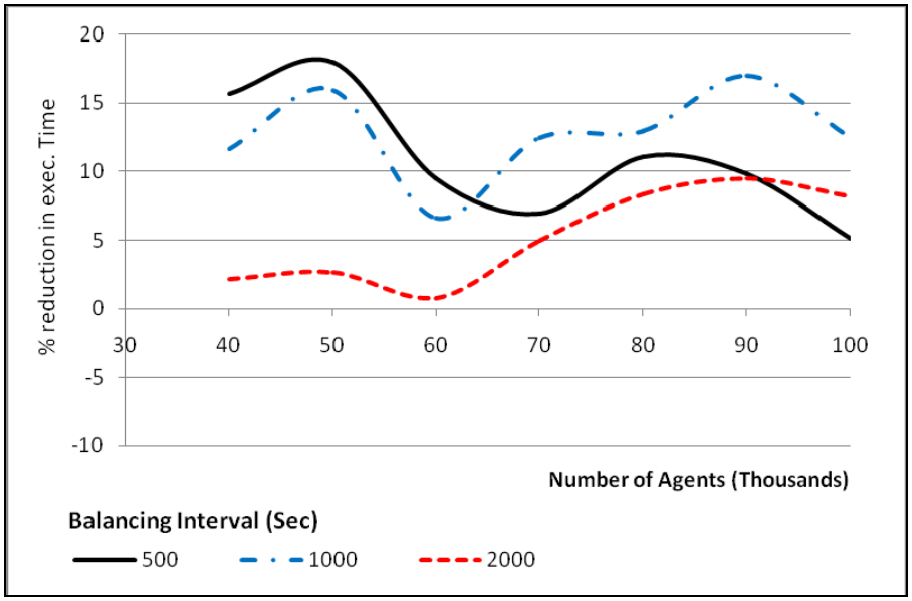


Figure 5. Performance comparison for different load balancing intervals

For small simulation sizes, the execution time is short and thus can only benefit from a shorter balancing interval. For larger simulation sizes, if the interval is too short, the overhead of file I/O, communication between user submission host and resources, load rebalancing computations, etc., add up to a significant magnitude, offsetting any possible gain.

Because smaller simulations execute for a shorter time, the preliminary prediction is accurate enough unless a major degradation of quality of service (such as failure of nodes, communication breakdown) occurs. Furthermore, in

small simulations, there is usually not enough data to train the prediction models for a better approximation.

It can be seen that the prediction error is not generally large (less than 15%) for this application. This is mainly due to the fact that there is virtually no coupling between tasks running at different nodes other than for load balancing purposes. There is no need to establish a central clock or a synchronization mechanism since each resource node can work asynchronously so long as it is given the parameter set it executes.

4.2. Large Scale Spatially Explicit Simulations

In these types of simulations, the agents assume well defined geographic space and act in and on it. Real world entities interact not only among themselves, but also with the environment they are situated in. Researchers are often interested to know how the population of certain species can, through complex dynamic processes, affect natural resources, climate, environment, etc.. A common modeling approach is to divide the physical region under investigation into rectangular cells or tiles identified by a geo-referenced coordinate system (usually in the form of a mesh). The size of the cells depends on the desired resolution. In some cases, the resolution is limited by the availability of adequate computing power. The cells possess contextually relevant attributes or active properties.

In a spatially-explicit agent based simulation, the agents modify the properties of the cells they inhabit or those in their surroundings. Such simulations have earned great interest in eco-system studies, land use management, crowd simulations, etc [31]. The simulations become complete if the tool used has a visualization component. Hence, many ABSS tools have GUI support for online visualization of evolutionary behavior.

One of the challenges of spatially explicit MABS is the size of memory required to map and store real world spatial data for processing, at the desired resolution level. Computational power needed to manipulate the spatial data as demanded by the simulation, within a reasonable time is also another challenge. There are cases where the problem demands simulating millions of agents and even larger number of cells. There is a limit in the size/ resolution of the physical space to be simulated without performance degradation. As

the ABSS tools are single threaded, they cannot handily be partitioned and executed in a distributed environment.

After studying spatially explicit Netlogo simulations, we proposed a distribution approach to tackle the problem. The simulation space is first partitioned into as many regions as the number of compute resources. In the NetLogo lingo, the simulated space is a 'world' made up of cells, called patches, which are agents of fixed size and location, arranged in a grid style. The movable entities called turtles are located with spatial coordinates [27].

Each resource executes an instance of NetLogo locally and simulates the region assigned to it. For this purpose, each resource has its own world that is generated and populated independently (using probability distribution functions). In order to get a simulation of the complete world, the simulation regions are merged. To make the merge as seamless as possible, neighbouring regions are allowed to have overlapping areas. Every host runs the simulation for one time step and then updates the states of turtles and patches. The updates at the boundaries require state information of adjacent cells from neighbor machines. At the end of the simulation run, updates of adjacent patches and turtles is passed to the neighbor over the network. To apply our load distribution approach, the following assumptions should hold:

- *Hosts with adjacent regions can communicate with each other directly to exchange information about their overlapping areas.*
- *If two hosts share adjacent cells, each side is responsible for updating only the cells on its side of the common border.*
- *The sizes (area) of the partition regions are based on the computational power of the resources they are assigned to.*

To avoid the situation where one simulation instance shares boundaries with more than two instances (like tiled cells), the partitions are taken lengthwise only.

Figure 6 shows two partitions, A and B, of a simulation world. In the figure, partition A shares borders with two others. Along the boundary, the host to which the partition is assigned is responsible for updating the states as shown. Compute resources hosting neighbouring partitions exchange boundary information through files. For large simulations, because one partition can occupy several hundred megabytes of memory space, the boundary data is as well large. We therefore compressed the data before it leaves the source host.

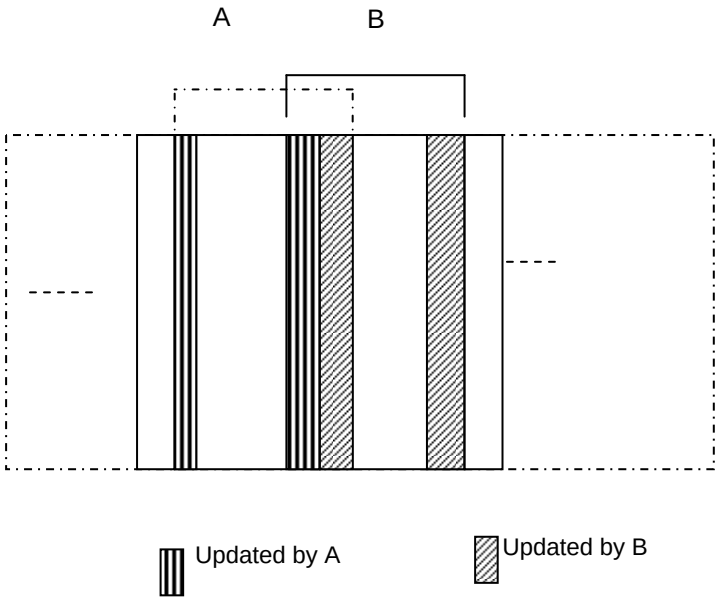


Figure 6. Partitioning of simulation space

The simulation overhead consists of the following components:

1. Saving the boundary states in a file and compressing it (using the *zip* utility, for example)
2. Sending the compressed file to the respective adjacent host
3. Decompressing received boundary data and updating the already existing states with the new data.

A substantial network overhead is also experienced during this process. The combined overhead could outweigh the reduction in simulation computation time.

Results

We performed experiments on the cluster and a Grid like environments with remote hosts serving each other their boundary data. We set up a simulation world consisting of 200,000 agents deployed uniformly in a tiled space of 1000X2000 points. We took measurements for different region overlap sizes and compared the speed up achieved.

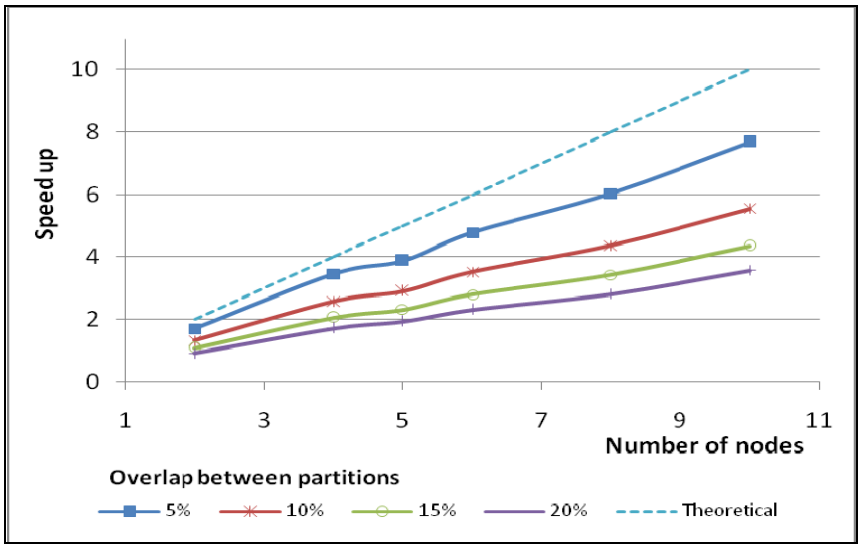


Figure 7a. Performance of a spatially explicit simulation on a cluster

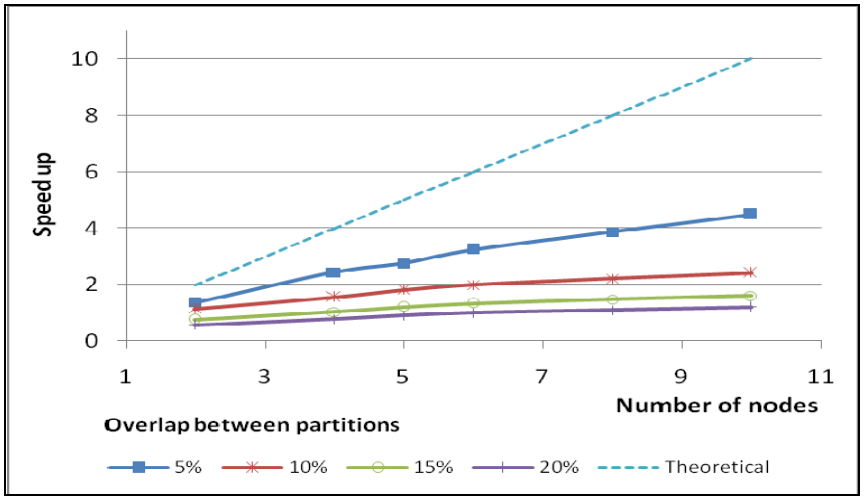


Figure 7b. Performance of a spatially explicit simulation on Grid

As can be observed by comparing Figures 7a and 7b, the speed up in the Grid environment is lower because the network latency overhead in the Grid is quite high. This implies that our approach is feasible only when the task is

distributed over several hosts. The partitions and hence the boundary sizes should be sufficiently small so that disk and network I/O overheads are reduced significantly.

To reduce the overhead, we introduced a modified concept of pipelining. It is possible to iterate through the cells in one partition, starting from the edges of the partition and progressing towards its center. Once the overlapping region at the boundaries is covered, a thread can be launched to carry out the disk and network operations required for saving, compressing and sending the common data to a neighbouring host. Another thread can also be used to listen to the arrival of data from adjacent hosts. We evaluated this approach on different hardware architectures. The gain on one CPU or single core machines is negligible. However, a substantial improvement could be achieved with multiprocessing machines as demonstrated with a remote SMP host shown in Figure 8.

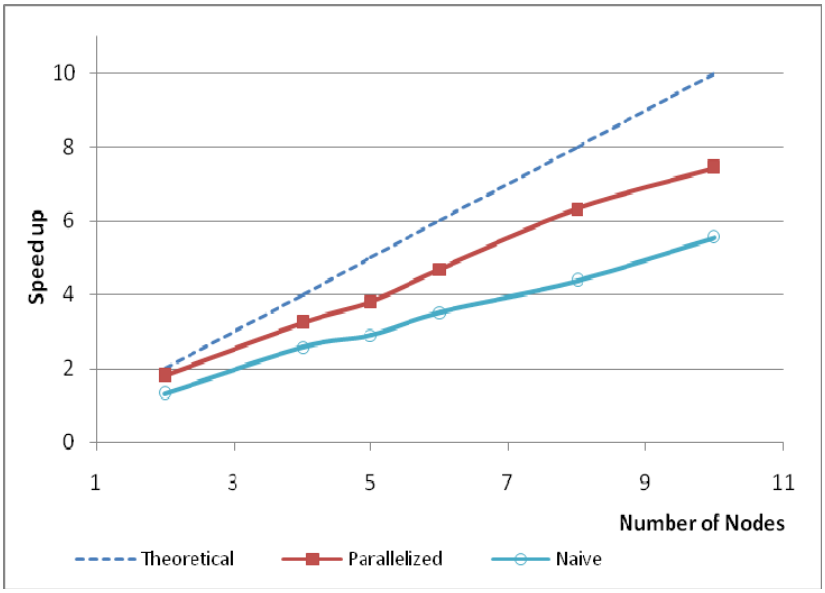


Figure 8. Performance improvement achieved by pipelining I/O operations with simulation execution.

The simulation application scales well as there is no need for a central clock or a global time synchronization mechanism. Only adjacent hosts need to synchronize among each other regardless of reference to other hosts.

However, a failure or I/O bottleneck can degrade performance or even bring the entire execution to standstill since the problem at one host propagates throughout to all participating nodes in succession.

5. MABS on Distributed MAS Platforms

In many real world systems, the entities defining the phenomena of interest exhibit higher level of intelligence. The entities are required to execute complex plans, be capable to learn, acquire and interpret data into knowledge, etc. The interactions among such entities also involve communications at the idea level, which may include multiple conversations, auctions, bids, offers and negotiations, to mention a few [5]. The communications take place not only among those agents in immediate neighbourhoods, but also with those apart contextually and/or geographically. Entities do not often have knowledge about their communication partners before hand and should get access to a dedicated information service to decide with whom to communicate about what. Although ABSS tools may be employed to simulate such systems, it is difficult to capture the dynamic interaction and intelligence of individual entities to the desired level of detail. Instead, MABS researchers employ multi-agent system (MAS) platforms which are specifically developed to address the above requirements. The prominent features of the popular MAS platforms include:

1. Support for complex interactions through standardized messaging services and interfaces;

2. Provision of white page and yellow page directory services to organize, register and discover agents as flexibly as possible;

3. Multi-threading to enable higher level of agent autonomy and even selectively replacing agents with their real world counter parts or by an independently running software. Each agent should therefore run in its own thread of control to meet this requirement.

4. Mobility of agents from one host to another.

Several MABS applications were deployed on MAS platforms to understand the behavior of systems in supply chain management, transportation and logistics, etc [34, 35].

5.1. Necessities for Distributed Execution

The last two features of MAS platforms explained above make them ideal for distributed simulations. However, each feature has a considerable overhead that causes serious performance bottlenecks in large-scale MABS applications. Although the large overheads associated with massive multi-threading are well known and thus not generally recommended, there is no other way to conveniently simulate certain real world systems.

One reason for executing a MABS application on several hosts is the limit set on the number of threads in a process or on a single machine. This limit depends mainly on the operating system, the run-time environment (for Java based applications) and the available memory. Some operating systems have only a 16-bit address space for threads, making 65535 as the upper limit for the number of threads that can be hosted on one machine. On 32-bit operating systems, the address space of one process is 2GB, out of which a certain amount is reserved for heap memory. In multi-threaded Java programs, stack memory is reserved for each thread upon creation (it is 512KB by default on Java1.5). This limits to the total number of thread instances to less than 4000. A simulation that captures the behavior of 20000 entities, for example, has to be executed on at least six machines.

Although MAS platforms offer a suitable deployment environment for MABS applications, they have the following limitations, owing to the fact that they were originally designed for other non-simulation applications.

1. There is no explicit central clock to synchronize the simulation execution. Failing to take care of this problem would result in inconsistency and violation of causality. It would therefore be difficult to ensure the repeatability of experimental findings. In a distributed environment, the problem is worse as maintaining coherent execution will be even more difficult.

2. There is no support to visualize the evolution of emergent phenomena in spatial context during the execution time. The GUI components that come with MAS platforms provide information mainly about the life cycle of agents, messages and directories that are of little interest to the simulation user.

3. The mathematical tools (if any) available in the MAS are not sufficient for simulation analysis. Statistical functions, optimization solvers, etc., are often needed in simulations.

The realization of the components needed to adapt MAS for MABS applications is the responsibility of simulation developers. The time step synchronization problem in distributed simulations is a well researched area as in [36, 37]. The type of synchronization approach used in a simulation affects the performance significantly as the data communication overhead can be overwhelming.

In our work, we studied the performance characteristics of MABS using the Java Agent Development framework (JADE) as our MAS platform. JADE is popular among the MABS community, particularly in the simulation of the systems explained earlier. The fact that it has extensible interfaces, good documentation and regular maintenance updates makes JADE an appropriate tool for other distributed applications as well [38].

JADE implements the basic agent platform services defined in the FIPA [32] specifications. These services themselves are implemented as agents: the agent management system (AMS) for life cycle management, the directory facilitator (DF) for agent directory look up services (white page and yellow page), and the agent communication channel (ACC) for message transport.

The JADE run-time environment is a Java container that hosts user created agents. It is launched in a Java virtual machine and provides access to the basic agent services through a set of APIs. Agents can send messages to each another according to the agent communication language (ACL) standard.

5.2. Workload model

As in any performance study, obtaining a representative workload is an essential part of the work. It is desired that the workload is parameterized in terms of the variables whose effect we need to quantify in the model. We identified the following variables to be captured in our workload model:

1. Simulation size (number of agents)
2. Complexity of agent behavior (task granularity)
3. Intensity of agent interaction (communication)
4. Computing resources (number of hosts)
5. Network latency

We designed the workload with the first four of the above parameters as the control variables. The effect of network latency is not quantitatively

captured in the performance model, but is observable qualitatively as we run our experiments in two network environments, cluster and Grid. A central clock is used to synchronize the simulation time at all hosts. In one simulation time step agents carry out their activities in two phases: first compute and then communicate. In the compute phase, all agents perform some amount of computational work corresponding to the task granularity parameter. In the communicate phase, the agents search the directory service (DF) to locate the agents they share information with. They will then update their knowledge of the environment by sending request messages to peers and receiving replies. The simulation is run for a pre-defined number of time steps, during which time the application is instrumented to collect performance data.

When the application is started, the JADE platform main container is launched on one of the hosts (head node of the cluster). On the other hosts, agent containers are launched and registered in the main container. In each container other than the main, the simulation agents are instantiated and execute their tasks as explained above. Each JADE agent is registered on the main container with a unique global address. It runs in its own container as an individual thread. JADE employs the LDAP protocol for the agent directory service.

A shell script is written to pass the workload parameters to the application so that they can be set at run-time. The task granularity (G) is the time duration the agent behaviour's code is executed in one compute phase. The communication parameter (r) is a measure of the proportion of outbound communication, i.e., the ratio of messages between agents residing on different hosts to the total number of messages transferred.

5.3. Results

Several measurements were taken for different simulation sizes in a cluster and Grid environment. The purpose of these experiments is to understand the performance characteristics of MABS applications deployed on MAS platforms. We investigated the communication behavior of the agents, the effect of the execution environment and other performance bottlenecks inherent to MAS.

The effects of agent task granularity and outbound communication in a cluster environment are shown in figures 9 and 10. The figures show the mean execution time for one simulation time step. We used 6 nodes on the cluster, with one of them assigned to host the platform main container.

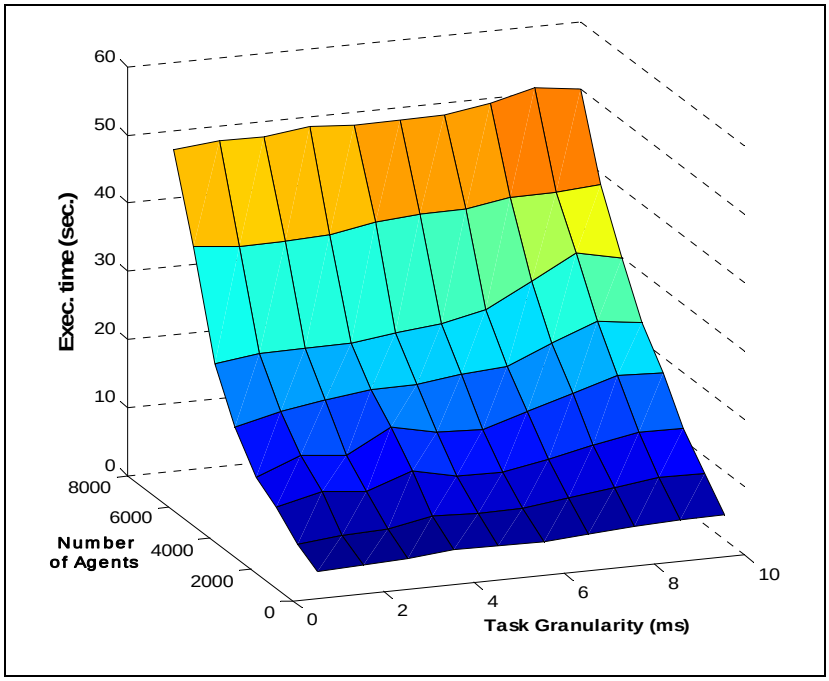


Figure 9. Workload execution behavior: variable granu-larity, fixed outbound communication ratio, r=40%.

It can be seen from Figure 9 that for large number of agents, the execution time increases exponentially regardless of the task granularity. We observed a similar behaviour when we run another set of experiments, fixing the task granularity to 1 msec, but with variable outbound communication ratio.

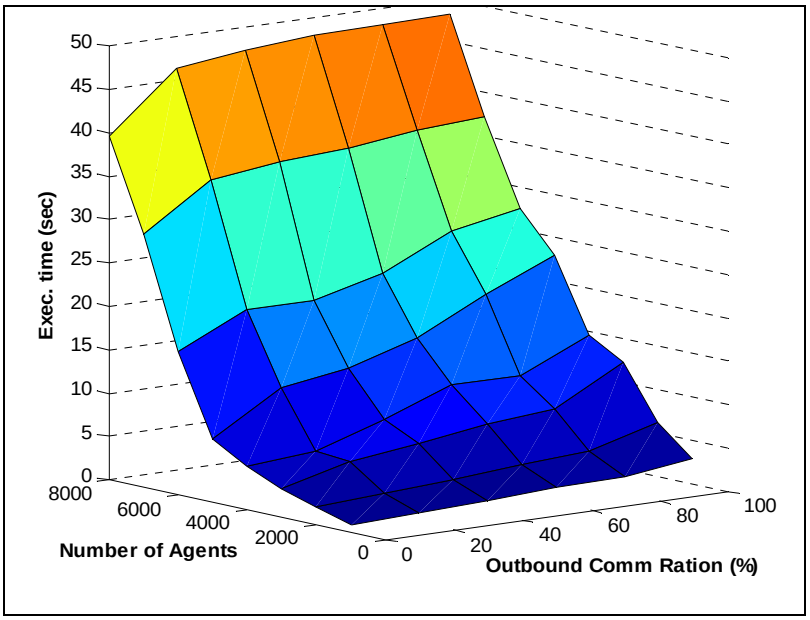


Fig 10. Workload execution behavior: variable outbound communication ratio, fixed granularity G = 1msec

From the workload execution plots it can be seen that the simulation does not scale for large number of agents even if the outbound communication and computational granularity are low. To understand the source of this performance bottleneck, we debugged and instrumented the platform code. We discovered that the response of the JADE Directory Service is very slow when the number of concurrent requests is high. The platform documentation states that this service is implemented based on the LDAP protocol which does not scale [39]. To overcome this problem, we introduced a directory caching mechanism. Instead of allowing each agent to access the central directory, the hosts retrieve a copy of the directory information and cache it in their local memory, availing to the agents deployed there. This reduces the directory access time drastically as can be seen by comparing the plots in Figure 11. The directory can be cached only once after all agents are launched (static cache) or is refreshed at the beginning of every simulation step (dynamic). The choice of the cache refresh rate depends on the agents' life cycle. In some simulations, the agents are instantiated at the beginning of the simulation and are terminated when the simulation ends. In such cases a static cache will be sufficient. In other cases, however, agents can be

instantiated or removed at any time in the course of the simulation. This requires dynamic updating of the cache.

We studied the performance of JADE based MABS in the Grid environment. The application was deployed on a cluster head node and other machines remotely connected over the Internet. One of the challenges we faced here is JADE's use of RMI for messaging. Although RMI can work without any problem in homogeneous clusters, it is not always possible to invoke it across domains due to security policies. On those nodes we accessed remotely, the execution time for the same type of workload was generally 60-80% higher than the cluster environment. This is clearly the contribution of network latency.

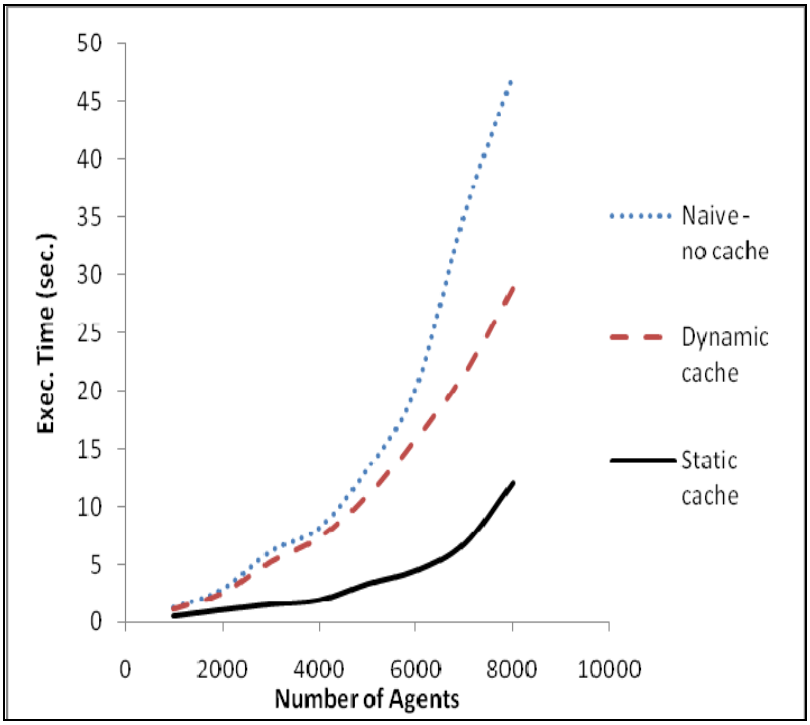


Figure 11. Performance improvement through caching the JADE agent directory contents (G=1ms, r=40%).

6. Web Services for MABS on the Grid

The last MABS application architecture studied is the web services model. The idea of using the web services as an enabling technology for distributed MABS is promoted due to two major reasons. The first reason is the proliferation of web services in implementation of distributed applications. For example, core components of the Globus toolkit (the software used for building Grid systems) such as execution management, data and resource management, security, information services and common run-time are all implemented as web services [22, 45].

The other reason for the adoption of this technology is the similarity between the existing practice of model composition for agent based simulation, and the concept of service oriented architecture (SOA). Researchers often prefer to build simulations by combining different ready-made template models and customizing them according to their specific needs [23]. It is argued that it the service composition and orchestration features of SOA can be exploited to build complex MABS out of prototype simulation models. Sanores and Pavon propose a framework for agent based social simulation in the Grid environment in [40].

In this work, we attempted to study the feasibility of an agent based simulation services platform and simulation services factory in the Grid environment. Prototypes of simulation agent models are implemented as publishable and discoverable entities on the Grid resources. The Globus Information Services component handles the task of registering these model components as services. The experiments with MAS platforms provided us with a great deal of insight on the architectural features of a distributed MABS. Task partitioning, application monitoring, dynamic load balancing and migration are essential requirements of the application. Each instance of an agent web service runs in its own thread of control, task distribution and load balancing can hence be handled more naturally. In accordance with the FIPA standard, the simulation web service template should have the following features (similar to MAS):

- Agent life-cycle management
- Standard messaging interface similar to ACL
- Directory information services
- Provision for customizing basic behavior of agents per the features of the real world entity it mimics.

6.1. Experiment

We set up a Globus Grid testbed for this experiment. All necessary components of the Globus toolkit (execution management, resource management, information services and security) were installed. We used the *simpleCA* certificate authority to implement security. The other components are all Java based. The simulation web services are deployed in the Apache Axis web service container. We used 6 Linux based machines as our Grid nodes in this experiment.

As a proof of concept, just two agent simulation service prototypes were designed and deployed on all Grid nodes. The user modifies the behavior class of a prototype according to how the entity in the real world would behave. The application is run in master-worker model, from a user host machine and creates agent instances through the *AgentFactoryService.* The master also contains agent life cycle management, directory and messaging services.

If the simulation consists of identical types of agents, load balancing becomes easy. Each Grid node hosts the same number of agent instances, equal to the simulation size divided by the number of Grid nodes. If the agents are heterogeneous, however, adaptive load balancing would be needed, as in Section 4.1.

The workload model, inter-agent communication and the input parameters applied in this experiment are similar to those we used in the MAS experiment in Section 5. However, communications involving cross host messages have to be routed through the master node as the Grid security policy does not allow direct link between hosts. Synchronization is maintained through a central clock on the master host.

We applied a similar procedure for data acquisition and analysis as in the MAS case. In our opinion, the most important accomplishments in this experiment are, understanding the performance characteristics of MABS applications implemented using web services, building performance prediction models for MABS workloads in a Grid environment.

6.2. Results

We measured the execution time of one simulation step by parameterizing the simulation size and outbound communication ratio for each execution

instance of the workload. Figure 12 shows the execution time of one simulation time for different simulation sizes and communication rates. Although in general, the execution time increases with the simulation time and outbound communication, the surface of the graph is a bit rugged. This is expected owing to the unpredictability of the Grid and the difficulty in precisely quantifying and modeling all factors involved. Although the workload has similar characteristics to the implementation on MAS platforms discussed in Section 5, for Web services based MABS have longer execution times. The following factors contribute to this high execution time:

1. Web services add significant overhead as data exchanged between service requester and provider are SOAP messages. The data is encoded in XML and wrapped in a SOAP envelope to be carried over http. For intense communication where every agent sends and receives messages asynchronously, the data conversion process introduces extra computation compared to JADE's messages.

2. The web services container (Apache-Axis) and application server are not optimized for large scale multi-threading. The server implements a thread pool to create web service instances of agents. For a large number of agents, the pool will be exhausted leading to building up of queues which degrades performance drastically.

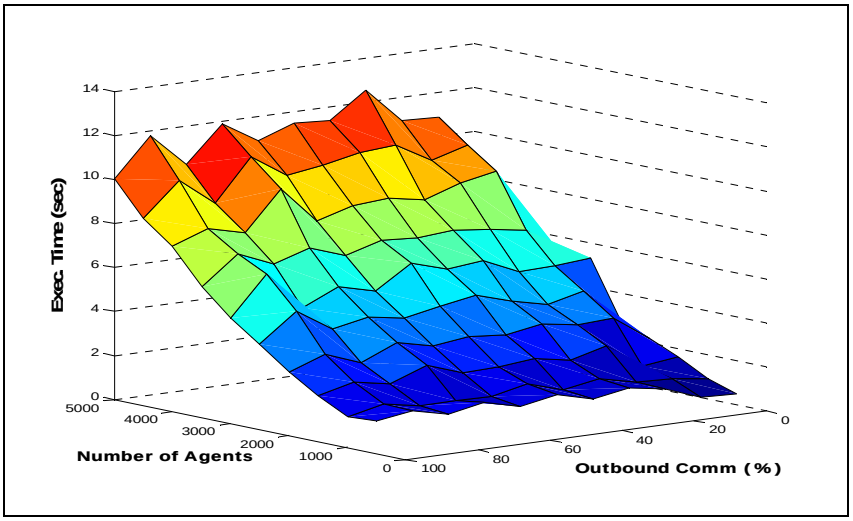


Fig 12. Execution behavior of a Grid MABS Workload on 5 Grid nodes: task granularity G = 1msec.

In one of our earlier works, we developed performance prediction models for MABS workloads in the Grid environment [2]. The models are obtained from run-time data of the workloads under different execution environments. They are used to identify sources of performance bottlenecks and make recommendations on optimal deployment scenarios.

In the prediction models, the simulation size, number of host nodes and task granularity are considered as independent variables. Agents are allowed to form groups by randomly picking peers from all hosts. Hence, the value of the outbound communication *r* is 50% for a simulation on 2 hosts, 67% on 3 hosts, *(M-1)/M* on M hosts, etc. The simulation execution time is a function of three quantities:

$$T_{exec} = f(N, M, G)$$

Where N = number of agents,
M = number of machines (hosts),
G = task granularity

Here too, as the execution behavior on the Grid is not predictable, it is difficult to obtain an accurate performance model as can be seen in Figure 13.

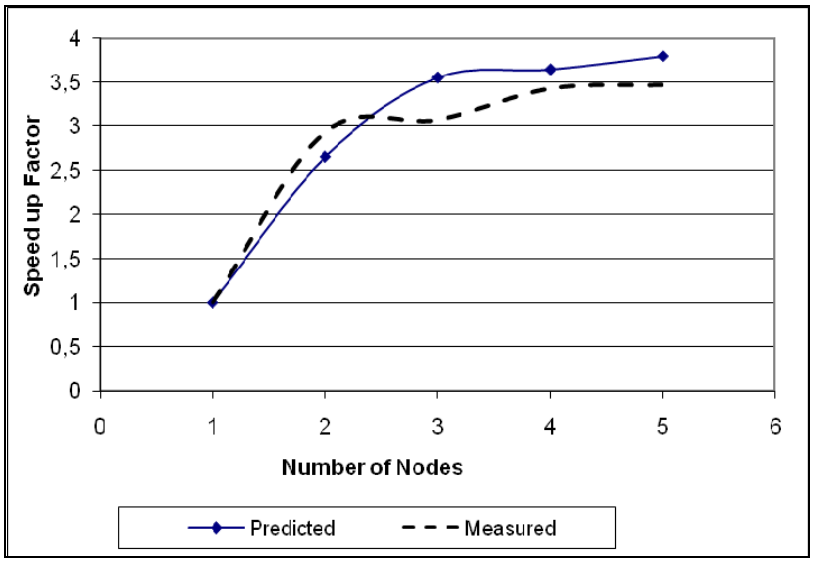


Figure 13a. Validation of performance prediction model with N=5000, G=5ms

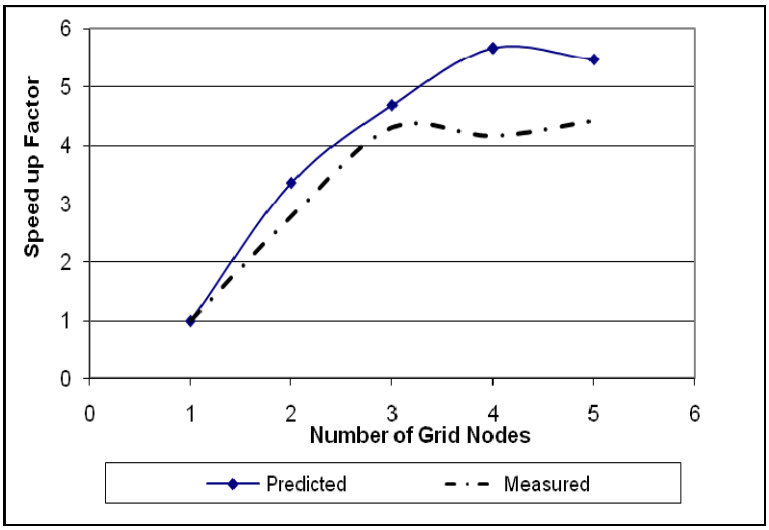


Figure 13b. Validation of performance prediction model with N=5000, G=1ms

We make two interesting observations from the plots in Figure 13. The first observation is that these applications do not scale well with increasing number of Grid hosts. The machine from which the application was launched becomes a bottleneck as all outbound communications have to pass through it owing to the Grid middleware requirements.

The second observation is, the speed up factor shows a super linear behavior, particularly in fine grained simulations. A closer performance analysis shows that when a large number of threads are launched on one node, the CPU spends relatively more time in kernel mode than in user mode. The operating system is forced to use the virtual memory due to resource limitations, and hence taking more time on swapping operations and other system calls than in the actual processing of the simulation. This overhead increases with simulation size and is independent of agent task granularity. Therefore, the proportion of the overhead relative to the useful execution time is more significant in fine grained simulations.

We attempted to build performance models for even finer granularity simulations. However, our measurement data show that it is more difficult to capture the system model accurately when the agent task granularity is a fraction of a millisecond (in the order of tens of micro seconds). This is

expected because factors that cannot be accurately predicted and quantified, such as thread context switching, network latency, time step synchronization and web services overheads constitute the major part of the execution time.

Web services introduce a considerable overhead during service instantiation and migration between resources. In our analysis, we did not explicitly include these and other sources of overhead such as web service response time, publishing and discovery of simulation services, etc. Although the web services solution for MABS does not have a satisfactory performance in the Grid environment, there are strong justifications to use this architecture as an approach to implementing distributed MABS.

The first reason is that it offers the technology to develop an extensible agent based simulation framework. Several simulations have similar types of entities. The governing behavioural logic of some entities are highly related as they are inspired by certain physical, social, biological, etc., phenomena. The same logic applied to model the behavioural dynamics of one system can be reused for another system in a different domain. Organizations, crowd, diffusion, ant formation, etc., are some of the behaviours exhibited in different real world systems [3]. If service components that capture the fundamentals of these behaviours can be publishable and discoverable, they can simplify the effort of simulation developers significantly.

The other reason for recommending the use of web services in MABS is the flexibility it offers to manage application execution in a distributed environment. Services can be deployed on any Grid host, and relocated as necessary, added or removed with demand. Task partitioning, performance monitoring, load balancing, task migration, etc., can be implemented in a very natural way as the bulk of Grid middleware is realized using this technology. A recent trend in distributed computing shows that service oriented architecture (SOA) is likely to be the model used to deploy and execute applications. Emerging applications developed for the Cloud use service oriented programming models and technologies.

7. Related work

An extensive amount of literature about agent based simulation tools and application case studies is available. Several application and resource

monitoring tools have been developed. One can also find important research work on performance modeling of distributed applications. However, we could not find any work combining the two areas in a comprehensive way. Our work can be considered as the first of its kind in applying performance modeling studies in the context of distributed MABS applications. We shall describe below the works we found relevant for our current research.

Davidsson et. al. [14] look at a higher level view of some MABS applications, such as agent organization and role, population/simulation size, agent behavior such as mobility, proactivity, dynamic instantiation, etc. They also study the tools and programming languages used to develop some of the MABS applications. An important finding relevant to this work is that currently most implementations are partial simulations despite the fact that full-scale simulations are required for reliable analysis and understanding of the real world behavior. We could further see that either new programs were written from scratch or desktop solutions were employed to develop the presented MABS applications.

Railsback et. al., [15] present a review of selected agent based simulation platforms developed in Java and Objective-C. They evaluated implementation details, programming efficiency, documentation, interfaces, and related productivity and usability attributes. In addition, they compared the execution speed of the selected tools by using a set of template models they named "StupidModel". This work gives a good insight into the workings of agent based platforms. It does, however, not address performance issues with respect to the execution environment. It was also explicitly indicated that they made their evaluations on a single desktop computer running the Windows XP operating system. The findings of this work contain important guidelines that helped us to narrow down the choice of agent based simulation platforms.

The PACE toolkit captures the performance model of an application and uses it for resource estimation, the model is used for estimation of resource requirements, scalability and scheduling decisions [41].

Ripeanu et. al. propose a performance modeling approach for the Cactus toolkit in the Grid environment [33]. This work gives an insight into development of mathematical prediction models for large scientific simulations.

Berman et. al. propose an application level scheduling (AppLeS) algorithm for heterogeneous Grid resources [29]. Their project focuses on task

partitioning and load balancing where it is not possible to control application execution with one global scheduler. As part of this project, they proposed a parameter sweep template (PST) framework and an adaptive scheduling algorithm [30].

8. Conclusions and Future Work

In this work, we investigated three approaches for distributed multi-agent based simulations. Our work also classifies MABS applications based on complexity of agents' behaviour and interactions among them. We studied publicly available MABS tools and applications to identify the architectural features that have influence on execution performance in a distributed environment. We also reviewed experimental works and surveys on efforts to develop distributed agent based simulations. The major contributions of this research are to help users of distributed MABS in the following:

- to determine suitable development, deployment and execution scenarios for MABS applications;
- to make recommendations on how the applications can be tuned, and adapt dynamically to changes in the underlying execution environment.

Some ABSS tools such as Netlogo can be parallelized as parameter sweep applications for certain types of simulation problems. Encouraging results that show the possibility of achieving scalable execution of parameter sweep ABSS on the Grid with adaptive scheduling and dynamic load balancing strategies were obtained. We demonstrated that simulations involving search based optimized solutions can be executed on a cluster or a Grid environment if the search space can be partitioned.

Our parameter sweep realization works well for exhaustive search though it may not always be optimal. However, in some cases, the parameter space is non contiguous and hence, not convenient for exhaustive search. We therefore observe that further investigation is needed to obtain heuristics that improve search efficiency in different situations. Another problem that needs to be addressed is concerned with search based optimizations. The parameters used for consecutive simulation iterations are estimated from the recent computations by applying genetic programming, gradient techniques, etc.,

techniques. In such cases, the next iteration's input parameters may be outside the scope assigned to the computing host. This causes idling of hosts, resulting in load imbalance.

A limitation of our proposed spatially explicit ABSS is in generating the simulation world. We observed that the layout of the combined simulation space is often different from our expectations. Care must be taken to make sure that all entities assume their intended spatial layout. One way to do this is, to generate the layout for the entire simulation space centrally, prior to partitioning and mapping the task pieces to the computing hosts. This is certainly a memory, I/O and network intensive operation as an image of the world must first be generated, partitioned and saved before dispatching to the respective execution resources.

Multi agent platforms can be used to simulate systems whose entities have a high level of complexity. Because many MAS platforms are parallel applications, they can be readily distributed. However, the intensity of interaction among entities and the computational granularity of agent complexity must be considered in deciding an optimal deployment configuration.

One of the ways to reduce the high communication overhead in MAS based simulations is to deploy agents that frequently interact with each other on the same machine. Unfortunately, the interaction information is often not available during initial agent deployment. We are currently studying how to overcome this problem by analyzing interaction patterns to obtain agent clusters for subsequent regrouping and redeployment to reduce outbound communications.

In simulations deployed on MAS platforms, the centralized management of agent communication and directory service affects application scalability significantly. Directory caching improves performance substantially as it places copies of directory information near the agents. Since LDAP is not efficient to manage a large number of agents, we recommend that better directory service organization techniques be sought.

We consider web services as an enabling technology to leverage MABS applications on the Grid. It facilitates resource monitoring and dynamic load balancing even on heterogeneous resources. Furthermore, the possibility of building MABS from distributed services opens a new frontier for simulation developers and modelers. Ready-made agent codes contributed by different

simulation programmers can be combined and customized to build and deploy MABS applications on geographically distributed computer resources.

In our future work, we intend to extend the web services implementation for sensor network research. We are currently working on a distributed sensor network platform to manage a large number of sensors. We shall use MABS to study and evaluate sensor net application scenarios. We shall utilize service composition and orchestration techniques to build complex MABS applications for sensor networks.

Regarding simulation time management, enforcing conservative synchronization of simulation time steps centrally degrades application performance significantly. This is particularly visible if the simulation agents' tasks are light-weight or have fine granularity. The possibility of employing other alternatives such as optimistic synchronization should be further investigated.

9. References

[1] G.M. Serugendo, "Engineering EmergentBehaviour: A Vision" *D. Hales et al. (Eds.) MABS 2003, Springer-Verlag, LNAI 2927, pp. 1-7, 2003*

[2] D. Mengistu, and P. Tröger, "Performance Optimi-zation for Multi-agent Based Simulation in Grid Environments". *8th IEEE International Symposium on Cluster Computing and the Grid. Lyon, France 2008.*

[3] B. J. L. Berry, L. D. Kiel, and E. Elliott "Adaptive agents, intelligence, and emergent human organization: Capturing complexity through agent-based modeling" *Proceedings of National Academy of Sciences of the USA, Vol. 99 Suppl 3, 2002*

[4] R. Axelrod, "Advancing the Art of Simulation in the Social Sciences" R. Conte, R.Hegselmann and P. Terna (eds.), Simulating Social Phenomena. LNEMS, Springer Berlin, 1997.

[5] C. Sierra, "Engineering Multi-Agent Systems as Electronic Institutions". *The European Journal for the Informatics Professional, Vol. 5 No. 4, 2004*

[6] P. Davidsson, J. Holmgren, J.A. Persson, L. Ramstedt, “Multi Agent Based Simulation of Transport Chains”. *Proc. of 7th Int’l Conf. on Autonomous Agents and Multiagent Systems, Padgham, et al, (eds.),* Portugal, 2008, pp. 1153-1160.

[7] M. Hare, P. Deadman, “Further towards a taxonomy of agent-based simulation models in environmental management”. *Mathematics and Computers in Simulation, Volume 64, No. 1, 2004, Pages 25-40*

[8] F. Wang, S.J. Turner and L. Wang. “Agent communication in distributed simulation”, Lecture Notes in Computer Science, Multi-Agent-Based Simulation III, 3415, 2005, 11-24

[9] Cioffi-Revilla, C. “Invariance and universality in social agent-based simulations”. *Proc. National Academy of Science USA, 99 (2002) Suppl. 3: 7314-6*

[10] K. Chmiel et. al. “Testing the efficiency of JADE agent platform”, *Proc. Int’l Symposium on Parallel and Distributed Computing*, Ireland, 2004, 49-56.

[11] Z. Xu, B.P. Miller, O. Naim, “Dynamic instru-mentation of threaded applications”. *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Georgia USA, 1999*

[12] X. Zhang, J.L. Freschl, J.M.Schopf. “A performance study of monitoring and information services for distributed systems”, *Proc. The 12th Int’l Symposium on High Performance Distributed Computing*. Seattle, USA 2003, 270-281.

[13] K.S. Perumalla, "Parallel and Distributed Simulation: Traditional Techniques and recent advances". *L.F. Perrone, F.P. Wieland, et al. eds.Proceedings of the 2006 Winter Simulation Conference.*

[14] P. Davidsson, et.al. “Applications of multi-agent based simulation” ”, *Lecture Notes in Computer Science, Multi-Agent-Based Simulation VI*, 5003 2008, 20-35

[15] S.F. Railsback and S.K.Jackson, "Agent-based Simulation Platforms: Review and Development Recommendations". *SAGE Publications Simulation Journal, Vol. 82, No. 9, 609-623 (2006)*

[16] M. Jang, G. Agha, "Adaptive Agent Allocation for Massively Multi-agent applications". *T.Ishida, et al. (eds.) MMAS 2004, LNAI 3446, pp.25-39, Springer-Verlag 2005.*

[17] T.A. Bergen-Hill, M.T. McMahon, B.F. Tivnan. "Our Summer with REPAST: Forging a Modeling and Simulation Foundation". *Agent2007, Conference on Complex Interaction and Social Emergence, USA, 2007.*

[18] M.B. Marietto et al. "Requirements Analysis of Agent-Based Simulation Platforms: State of the Art and New Prospects". *Multi-Agent-Based-Simulation II Springer Berlin, LNCS Vol. 2581, pp183-201 2003*

[19] R. Tobias and C. Hofmann "Evaluation of free Java-libraries for social scientific agent based simulation" *Journal of Artificial Societies and Social Simulation 7:1 (2004) http://jasss.soc.surrey.ac.uk/7/1/6.html visited October 2009.*

[20] I.J. Timm, D. Pawlaszczyk. "Large scale multi-agent simulation on the Grid". *IEEE International Symposium on Cluster Computing and the Grid 2005*

[21] I. Foster, H. Kishimoto, et al. "The Open Grid Services Architecture, Version 1.0," *Open Grid Services Architecture WG, Global Grid Forum, 2005.*

[22] M. Atkinson, D. DeRoure et al. "Web service grids: an evolutionary approach". *Journal of Concurrency and Computation Practice Vol. 17 pp. 377–389, 2005.*

[23] N. Gilbert and K.G. Troitzsch, "Simulation for the Social Scientist". *Buckingham, U.K., Open University Press, 1996.*

[24] F.P. Bellifemine, A. Poggi, and G. Rimassa, "JADE A White Paper". *http://jade.tilab.com/-papers/2003/ WhitePaperJADEEXP.pdf, Visited March 2008.*

[25] S. Luke et al., "MASON: A Multi-agent Simulation Environment". *SAGE Publications Simulation Journal, Vol. 81, No. 7, 517-527 (2005)*

[26] REPAST Agent Simulation Toolkit. *http://repast.sourceforge.net/ Visited January 2009.*

[27] The Net Logo Home Page. *http://ccl.northwestern.edu/netlogo/ Visited January 2009.*

[28] Theodoropoulos G, et. al. “Large scale agent-based simulation on the Grid”, *Proc. 6th International Symposium on Cluster Computing and the Grid*, 2(1), Singapore, 2006.

[29] Berman et.al. “Application-Level Scheduling on Distributed Heterogeneous Networks”, *Proceedings of Super computing, ACM/IEEE, USA, 1996.*

[30] H. Casanova et al., "The AppLeS parameter sweep template: user-level middleware for the Grid*". IOS Press Journal of Scientific Programming, Vol. 8, No. 3, pp 111.126,2000*

[31] Kota, R., Bansal, V., and Karlapalem K. “System issues in crowd simulation using massively multi-agent systems” *Proceedings of Workshop on Massively Multi Agent Systems. Kyoto, Japan. 2006*

[32] The Foundation of Intelligent Physical Agents. http://www.fipa.org, visited August 2009.

[33] M. Ripeanu, A. Iamnitchi, I. Foster. ”Cactus application: performance prediction in Grid environments”, *Lecture Notes in Computer Science 2150, 2001, 807-816*

[34] J. Holmgren, et. al. “An Agent Based Simulator for Production and Transportation of Products”, 11th World Conference on Transport Research. Berkeley, USA. 2007

[35] N. Julka1, R. Srinivasan, and I. Karimi, "Agent-based supply chain management—1: framework". *Computers & Chemical Engineering, Elsevier, Vol. 26, No. 12, pp. 1755-1769, 2002.*

[36] D.R. Jefferson, “Virtual time” *ACM Trans. on Prog. Languages and Systems 7(3) 1985. 404–425.*

[37] J. Misra, “Distributed Discrete Event Simulation” *ACM Transactions on Computing Surveys, 18(11), 1986. 39-65.*

[38] G. Vitaglione, F.Quarta and E. Cortese, “Scalability and Performance of JADE Message Transport System”, *Proc. AAMAS Workshop*, Bologna, Italy, 2002

[39] X. Wang, H. Schulzrinne et al. ”Measurement and Analysis of LDAP Performance”. *IEEE/ACM Trans-actions on Networking Vol. 16, No. 1, 2008*

[40] C. Sansores and J. Pavon, "A framework for agent based social simulation". *Second European Workshop on Multi-Agent Systems, Barcelona, Spain, 2004*

[41] S.A. Jarvis, D.P. Spooner, H.N.L.C. Keung, G.R. Nudd. "Performance prediction and its use in parallel and distributed computing systems," *Proc. 17th International Symposium on Parallel and Distributed Processing,* Washington, DC, USA 2003

[42] K. Jurasovic, G. Jezic, M. Kusek, "A performance analysis of multi-agent systems", *Int'l Transactions on Systems Science and Applications, 1(4), 2006.*

[43] E. Bonabeau, "Agent-based modeling: Methods and techniques for simulating human systems". *Proceedings of National Academy of Sciences of the USA, Vol. 99 Suppl 3, 2002 pp. 7280-7287*

[44] E. J. Hastings, J. Mesit, R.K. Guha. "Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing" Proceedings of the 2005 Summer Computer Simulation Conference, Vol. 37, No. 4, pp 9-17, 2005.

[45] I. Foster, "Globus Toolkit Version 4: Software for Service Oriented Systems" *Springer, Journal of Computer Science and Technology Vol. 21, No. 4*, 2006, pp. 513-52

NINE

Migration Heuristics and Performance Models for Distributed Multi-Agent Based Simulations

Dawit Mengistu[1], Paul Davidsson[1], Lars Lundberg[1], Peter Tröger[2]

[1]*School of Computing, Blekinge Institute of Technology,*
371 40 Karlskrona, Sweden
E-mail: {dawit.mengistu, paul.davidsson, lars.lundberg}@bth.se

[2]Operating Systems and Middleware Group
Hasso-Plattner-Institut, University of Potsdam 14482 Potsdam, Germany
E-mail: peter.troeger@hpi.uni-potsdam.de

Abstract

Multi-agent based simulation (MABS) is an emerging computational tool used for studying complex real world systems. Many MABS applications are very large and are not suitable for execution on a single computer. However, they usually have performance and scalability problems in a distributed computing environment due to their high communication-to-computation ratio. Existing communication optimization approaches for distributed applications are often ineffective in overcoming this problem due to the heavy overhead they incur. In this paper, we present an efficient agent migration algorithm to reduce communication overhead in distributed MABS applications. Our algorithm employs heuristics to identify agent interaction graphs and communication clusters and proposes a migration scheme to reduce the communication

burden on the execution environment. We use an online performance prediction model to determine the effectiveness of the proposed scheme to reduce the overhead satisfactorily. The algorithm computes the migration scheme iteratively, yielding considerable optimization in performance. The main strength of the algorithm is that it does not incur significant overhead because it mainly uses the idle-times of the simulation coming from the communication overhead. Experiments show that the proposed algorithm can lead to a substantial reduction in communication overhead, and hence, an improvement in application runtime.

Keywords: *Agent based Simulation, MABS, Agent Migration, Algorithm, Performance prediction, Performance optimization,*

1 Introduction

Multi-agent based simulation (MABS) is rapidly developing as an alternative to traditional simulation methods for studying complex real world systems comprised of autonomous and interacting entities. The capacity of computing resources required to run a MABS application needed to study a social system depends on the complexity and size of the system to be studied. The size is usually derived from the number of entities being modeled and simulated [1]. The complexity is a measure of the computational granularity of the task of an individual entity, which is the corresponding volume of program code needed to capture the role of the entities. In some studies, the invariance of the findings drawn from MABS cannot be assured if the conclusions depend on the size of the simulation [16]. For this reason, it is often required to undertake a full-scale simulation by modeling the behaviour of all entities of the system and the interactions among them. A MABS of such a scale often cannot be carried out in reasonable time on a single computer and requires a huge amount of computing resources.

Optimization of MABS applications in distributed environment is critical because parallelized MABS applications generally have a well-known problem of high communication-to-computation ratio [3]. This problem is worse in loosely coupled systems such as clusters and Grid where data exchange take place over network paths of high communication latency. Accordingly, MABS applications are known for their very poor utilization of the resources they are executed on [15]. Without application level optimizations, the performance loss due to communication bottlenecks would far outweigh the expected benefits from distributed processing. It would therefore be impossible to achieve application scalability and the anticipated gain in execution speed.

One obvious approach to reduce communication overhead is to deploy the MABS agents in such a way that most of the inter-agent communication takes place among agents allocated to the same node. However, this approach requires prior explicit knowledge of the agent interaction graph, while in reality this graph is often not available before hand, and even if there is one, it changes as the simulation progresses. Regrouping and reallocation of agents involves studying their self-organization and group formation behaviour and, obtaining the communication graph on-the-fly to make migration decisions

dynamically. It needs to be considered that migrating agents is particularly a computationally expensive task in distributed execution environments.

This research employs performance prediction models to estimate the expected gains from agent migration and iteratively evaluates performance improvement schemes to determine an optimal one under prevailing operating conditions. It extends the modeling approaches in other Grid based scientific applications to include special features of agent-based simulations. As we shall show, the models provide better insights into communication optimization in distributed MABS and facilitate dynamic and online agent reallocation decisions.

The achievable reduction in communication overhead depends on the inter-agent interaction behavior in a given MABS. Some systems are characterized by more intensive communication among their entities. The interaction structure can be fluid with no permanent group formation patterns, or it can be static, random or well defined with large or small group sizes. An efficient strategy to analyze the agent interaction graph is therefore needed to decide on the viability of agent migration to relocate agents according to their grouping patterns. Furthermore, it should be possible to quantify the ensuing overhead versus the expected improvement to decide whether migration brings about a meaningful improvement in performance. A mathematical model characterizing the execution performance of MABS applications is therefore needed.

In this paper, we propose a novel approach to analyze agent-interaction graphs and to use performance models to make agent migration and message aggregation decisions. The contributions of this work are two-fold: first, we present a low overhead distributed algorithm to generate agent-interaction graph and a heuristics that uses this graph to produce an appropriate migration scheme for reduction of communication overheads. The efficiency of this algorithm depends on the fact that it amortizes its overhead over time and space. The second contribution is the development of prediction models to characterize the performance of MABS applications through which informed decisions on the use of migration strategies can be made.

The rest of this paper is organized as follows: a brief overview of related work is presented, followed by the methodology applied in this work. We then explain the design of our experimental workloads and develop their mathematical models. Following this, we discuss the distributed algorithm for generating agent-interaction graph and develop the migration heuristics

algorithm. In the Results section, we present the performance of our migration algorithm, the prediction models and evaluation of our performance optimization strategies. We finally discuss the important findings of this research and conclude the paper by citing directions of future work.

2. Related work

A significant work has been carried out in the areas of performance prediction, communication overhead reduction, and load balancing of distributed applications. Several tools were developed for application and resource monitoring, and performance modeling of large-scale scientific applications. However, performance studies of MABS applications are at an early stage as MABS is a relatively recent entry into the distributed computing arena. Existing prediction models assume applications with relatively coarser task granularity and cannot be directly utilized in the typically fine-grained MABS applications. Moreover, these models expect predictability in inter-task communication and execution flow, features which are characteristically absent in MABS.

The review of relevant literature in distributed agent based simulations shows that reduction of communication overhead through agent migration improves the execution performance of MABS[2, 18, 21]. To address this problem, we proposed an agent migration algorithm in one of our earlier works. The algorithm is based on a heuristics which improves the communication performance of MABS in a Grid environment [3]. It anticipates performance gains from migration considering that the simulation will thereafter run with a lower communication overhead for a sufficiently long time, eventually offsetting the migration cost. However, this work does not show how the optimality of the heuristics can be predicted before it is implemented.

Ripeanu et al. propose a performance prediction approach for the Cactus toolkit in the Grid environment [7]. This work gives a useful insight into the mathematical prediction models for large scientific simulations. As mentioned above, these models were not sufficient for MABS as they do not adequately model the execution behavior of MABS.

In our previous research on distributed MABS, we developed mathematical models for application performance prediction and for evaluating different performance optimization approaches [3, 5, 15]. In our current research, we propose an algorithm that progressively builds a migration scheme without incurring significant overhead on the application execution time. Performance prediction models are used to evaluate the feasibility of the migration scheme and to iteratively obtain an optimal one.

3. Approach

MABS applications are not usually developed from scratch. They are deployed on either a multi-agent system (MAS) platform, or an agent based modeling and simulation (ABMS) tool. Our choice of platform in this work is the Java agent development framework (JADE) platform, an open-source software popular in the agent community [12]. JADE provides extensible APIs for agent communication tasks that can be easily modified or re-implemented according to the needs of the user. It offers a good experimentation platform with low level functionalities to enable measurement, analysis and improvement of communication performance. It also allows the insertion of instrumentation code at desired points to make performance measurements, and predictions from within the simulation program. Furthermore, the collection and analysis of communication meta data, generation of interaction graph and migration heuristics can be performed with a relative ease.

To investigate the communication behaviour of MABS applications in a realistic setting, we ran the JADE-based experiments in both homogeneous and heterogeneous environments. We designed two representative workloads containing the essential features of typical MABS applications. The communication models follow conventional agent interaction topologies based on peer-to-peer and hierarchical structures [1].

MABS applications are not suited for distributed execution due to the overwhelming communication traffic they generate. We introduced some optimization features to minimize the need to communicate with the node hosting the main platform (central node). One of these features is to keep copies of the white and yellow page directory information at every node to minimize directory requests and to handle inter-node messages without

involving the node hosting the main platform [14]. The most important function of the central node under normal conditions is then maintaining time step synchronization globally. We developed a novel approach to iteratively update agent interaction graphs and migration schemes at computer nodes so that a common view of the entire simulation is achieved across all nodes.

4. Workload and Agent Communication Model

We designed two representative workloads containing the essential features of typical MABS applications. The communication models are based on the two common agent interaction topologies of existing MABS workloads [1]: peer-to-peer and hierarchical. For the peer-to-peer workload, a group formation problem in social networks [23] was considered. For the hierarchical case, we considered a problem in the transportation and logistics domain where the simulated system is modeled as a multi-level organization [17]. Both models rely on a common mathematical model for representing their properties.

4.1. Mathematical model of the workload

We consider a simulation size of N agents deployed on M connected machines such as nodes of a compute cluster or Grid. We further assume a time-driven execution of the MABS workload. Thus, if the simulation is run for n_{ts} time steps, the total execution time will be

$$T_{sim} = p\, n_{ts} \qquad (1)$$

Where p is the total duration of one time step.

Each time step has two phases, a computation phase and a communication phase. In the computation phase, a simulation agent executes program code corresponding to the tasks of the real world entity it simulates. In the communication phase, it may send (and receive) one or more messages to (and from) other agents. To maintain causality, the two phases are synchronized centrally and clearly separated so that no agent falls behind or advance ahead of the rest of the group. It then follows that

$$p = t_{comp} + t_{comm} + t_{sync} \qquad (2)$$

where t_{comp} and t_{comm} are the computation and communication times in one simulation time step respectively, and t_{sync} is the idle time that the agents spend waiting for the central synchronization signal to advance to the next time step. We further define

N = the simulation size (number of agents),

M = number of computing nodes

G= the average length of an agent's task in the computation phase (also called task granularity),

Assuming all nodes host the same number of agents and neglecting the thread context switching overhead (too small compared with G), we have the average computation time

$$t_{comp} = N\,G/M \qquad (3)$$

The number of messages sent over the network expressed as a percentage of the overall inter-agent messages is defined as the outbound communication ratio, r. If the initial agent allocation is carried out in a random manner, then assuming that $N >> M$,

$$r = 1 - (1/M) \qquad (4)$$

In simulations where a priori information about the behaviour of the agents is available, *smart* load generators may be used to apply informed initial allocations so that the actual r is minimized.

4.2 Agent Communication Model

To investigate the properties of the communication model, we consider a simulation where agent behavior such as its role and grouping was determined a-priori. Agent deployment is carried out in such a way that the number of outbound messages (the ones destined to agents whose peer live on a different machine than the originator of the message) does not exceed a given value of r.

Because agents in JADE are executed as threads, they stay in a wait mode until they receive a reply. Inbound messages (within the same machine and the same JADE container) are delivered using event passing by cloning the

message, and the new object reference is passed to the receiver [12]. Outbound messages are routed based on the replicated directory information and delivered using RMI.

Figure 1 Waiting time for a reply on a cluster. Each node hosts 500 agents.

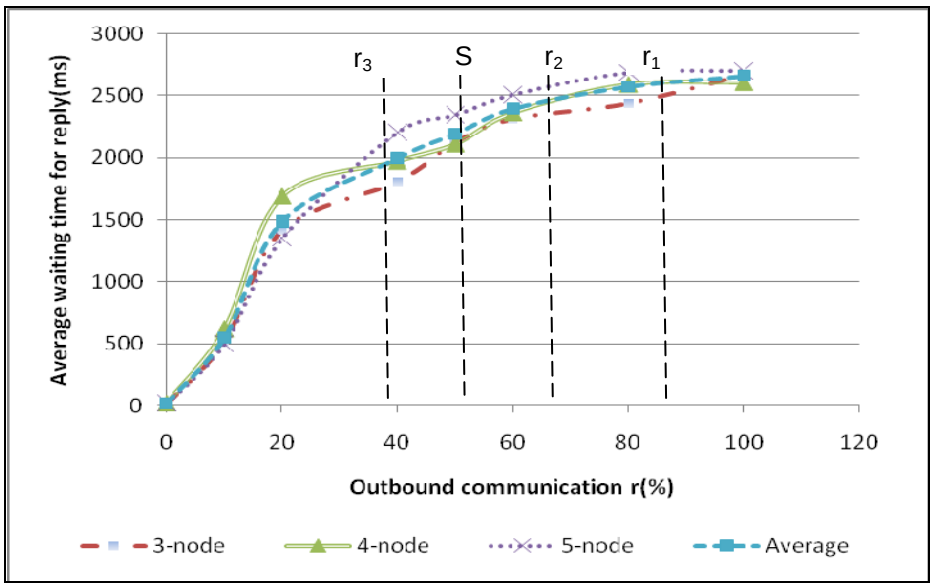


Figure 1 shows the resulting communication behavior for this experiment. The vertical axis shows the average waiting time of an agent for a reply from a peer. It can be observed from the figure that there is a sharp rise in the waiting time when the outbound rate is increased. The slope will become gentler thereafter. To explain this behavior, we consider the following mathematical model for a distributed MABS.

Let all agents deployed on a host send messages to one of their peers located on another host *(r=1)*. We define
t_{wait} = waiting time.
N_i = the number of agents on machine i.
a_{ij} = agent j deployed on node i
L = network latency
G_{ij} = amount of work performed in the role task by agent j at node i
t_s = time to process the message on the sender side, which is the cost of remote invocation [13].

t_r = time to process the message on the receiver side.

The reply waiting time by an agent $a_{ij,}$ for a message it sent to agent a_{mn} $(i \neq m)$, assuming that all machines start the simulation at the same time and, ignoring context switching and related overheads is

$$t_{wait} \geq 2(L + t_s + t_r) \quad (5)$$

(Equality means that the receiver gets the message immediately after it arrives on its host, sends a reply instantly, and the original sender receives the reply in the same way).

The worst waiting time occurs when the sender agent thread is executed first and the receiver is executed last on their respective hosts. Similarly, we expect that all the agents on both machines are also sending (and receiving) messages. Because MABS applications are normally fine grained, the amount of an agent's role task computation between successive communication instants is small. The upper bound for wait time will be dominated by the network latency. Thus,

$$t_{wait} \leq 2N_jL \quad (6)$$

If the outbound rate r<1,(only some agents communicate with peers at different hosts) then

$$t_{wait} \leq (2N_jL)r \quad (7)$$

The computational work performed by a single CPU machine to complete one cycle of role task execution and messaging (sending and receiving a reply) is

$$t_{comp} = \sum_{i=1}^{N} G_i + N_j(t_s + t_r) \quad (8)$$

The right term is the contribution of network latency. In general, for $r<1$,

$$t_{comp} = \sum_{i=1}^{N} G_i + N_j(t_s + t_r)r \quad (9)$$

If $N_i = N$, $G_i = G$, as in equation (3), and $t_r = t_s = t_{rs}$, we can write:

$$t_{comp} = NG + 2Nrt_{rs} \quad (10)$$

The machine will be 100% busy (no idling) if the wait time does not exceed the computation time, i.e.,

$$t_{wait} \leq t_{comp} \quad (11)$$

Assuming G << L,
Zero machine idle time condition can be achieved (i.e., $t_{sync} = 0$ in equation (2)) if,

$$r < G/2(L - 2t_{rs}) \qquad (12)$$

Several factors contribute for high overhead and thus waiting time when r is large. There will be a lot of resource contention and collision, and a very high overhead originating from the unpredictability of Java thread scheduling and management for such a large number of active threads. Coupled with network congestion and collisions, it increases the waiting time almost quadratically for large simulation sizes until the network saturates. After a sharp increase in the communication overhead, a point will be reached where the contribution of the other overheads dominates (line S on the graph in figure 1). Beyond this point, the increase will be less sharp, even if r is higher.

As an example, a MABS application which is currently running with $r=r_1$ achieves little performance gain if agent regrouping through migration reduces the outbound rate to r_2. Although the reduction in r is substantial, it is not accompanied by a proportional improvement in waiting time. A regrouping which can bring the r down to r_3 or less is desired to achieve a significant performance gain. However, this is often very difficult as if the interaction graph is dense.

An important finding of this experiment is that, the reduction of outbound communication will not always result in a meaningful performance gain. Therefore, before any migration decision is made, its cost and the expected reduction in communication overhead must be considered.

5. Agent Interaction Graph for Migration Decisions

Inter-agent communication is a defining characteristic in MABS applications performance. It depends on the agent interaction behavior and the architecture of the MABS. Analyzing and understanding this characteristic is a pre-requisite to reduce communication overhead.

5.1. Agent interaction behaviour

If the peer or group relationship of simulation agents is known at the time of launching the application, it will be possible to deploy the agents in accordance with their communication pattern with those agents with a higher interaction frequency being co-located. In such cases, a static load generator handles the optimized deployment and launching of agents and there would practically be no need to load balancing from a communication perspective.

In other cases, the interaction behaviour is vaguely known and in fact, is an essential outcome of the simulation. The inter-agent communication graph has to be determined by studying the interaction patterns of individual agents and forming aggregate structures. At the time of launching the simulation, the agents are deployed randomly on the available compute nodes. In summary, the interaction behaviour can be defined in the following ways:

Static vs dynamic: in the static case, agents interact with the same peers throughout the simulation. In the dynamic case, the agents often communicate with different peers and it is difficult to generate a fixed interaction structure for them. Graph clustering techniques can be used to generate the interaction graph for dynamic groupings [22], but this is not feasible for our case as they are time consuming and the result is likely to be of little use.

Clustered vs cluttered: in the former case, agents tend to form grouping around a certain number of agents. In the later case, there is no group formation. While it is easier to apply migration heuristics to cluttered interactions, the same may not be true for clustered ones if the size of the clusters spans beyond the capacity of one machine or there exist multiple clusters that cannot evenly fit on several machines.

5.2. Application architecture

The interaction pattern also depends on the organization of the real words entities modeled in the simulation. Accordingly, it is possible to have the following architectures:

Peer-to-peer: no pattern of organization is observable if the entities are decentralized or not well organized as in cooperative problem solving.

Hierarchical: the agents form an organization structure reminiscent of a business entity. The hierarchy could be vertical only (no agent can have more than one superior) although in some cases, diagonal interaction may be allowed (agents can report to one or more superiors).

Hybrid: a combination of the above two.

To optimize the necessary migration decisions, a priori knowledge of the architecture and interaction behaviour would be very helpful. We will now show possible approaches for interaction graph computation.

5.3. Discovering the agent interaction graph and agent clustering

Agent regrouping and redeployment through migration involves three significant tasks. The first and major step is to generate a global agent interaction graph. This requires studying the communication characteristics of individual agents by collecting message logs from individual agents' message boxes. The next task is to identify agent clusters from the graph and define agent groups according to the cluster formations. Finally, migration schemes are prepared in such that minimizes the number of movements across nodes if agents happen to already be not co-located with their peers in the agent cluster they belong to.

For large simulations with thousands of agents deployed on each machine, the generation of interaction graph and clustering is an overwhelming task. Because popular graph clustering techniques require that the meta-data of message logs to be processed centrally, a lot of overhead will be incurred in transferring this data from all machines to a central node. The edge matrix of the resulting graph is also very large. Cluster analysis and statistical physics tools will have to be employed to analyze the data and determine the clusters [22]. This task, if at all there is sufficient resource on the processing node, takes a large amount of time. Ironically, our attempt with 20,000 agents shows that, the useful simulation time pales against the time needed to perform cluster analysis just once in the middle of the simulation execution. It does not worth to apply a graph clustering algorithm that consumes more time than the simulation itself [24].

In order to deal with this complexity problem, we developed a heuristic which progressively generates a distributed agent cluster graph and produces a

sub optimal migration scheme to redeploy the agents accordingly. The efficiency of our heuristic emanates from the following major factors:

1. Most of the processing overhead for the clustering task is amortized throughout the execution of the simulation. At each node, there is a substantial idle time which agents spend waiting for messages or state updates from other nodes. It takes just a small fraction of this idle time to generate the agent clusters locally and make migration decisions.
2. Since our implementation of the heuristic is decentralized, the entire clustering operation is executed in a distributed manner and no single node is over loaded with this task.

From an implementation perspective, we launch a special migration agent on each node is responsible for analyzing the interaction pattern of the agents. Each simulation agent uses a data structure to record its inbound and outbound adjacency lists. When it sends a request message or receives one from others, it updates its list. When communication is established with an agent on a different host for the first time, the migration agent creates a counter with the host number h ($0 \leq h \leq M-1$). For successive communications with the same host, the counter is simply incremented to track the number of peers on that host.

To identify the first set of agents to move, the migration agent parses through the inbound peer data. If an agent has no entry in this data, it would be a good candidate for migration and can be moved to the host where it has majority of its peers. Successive iterations would be conducted to identify those agents whose inbound adjacency list is small.

Let

$mig(i)$ = migration agent at host i

$p_{av}(i,j)$ = average number of peers for agent a_{ij}

Let $p_m(j)$ = number of peers hosted on machine m

$$\text{Thus,} \quad \sum_{m=1}^{M} p_m(j) = M p_{av}(i,j)$$

If $p_i(j) < p_{av}(i,j)/M$, then it means that this agent has more outbound peers and is a potential candidate for migration. The migration agent then decides an appropriate host for it (the highest $p_m(j)$). The advantage of this algorithm is

that $p_m(j)$ is updated iteratively by a_{ij} itself while the simulation is running. Therefore, the migration agent will have less work to do.

The migration agent prepares its schedule from these data and dispatches it to all migration agents running on the other hosts. This schedule contains the list of agents to be relocated and their future hosts' identities. There is, however the following drawback with this approach:

Let two a_{ij} and a_{pq} be peers with no entry in the inbound adjacency data structure.

If *mig(i)* decides to relocate a_{ij} to host p and *mig(p)* decides to relocate a_{pq} to host i, the two agents are still not collocated. The same things can possibly happen to all relocated agents and thus, there are possibilities that no improvement is achieved.

To overcome this problem, we modified the algorithm in such a way that after a certain *mig(k)* dispatches its schedule, all recipients revise their schedule based on the latest information. Another challenge here is, how to establish the order in which individual migration agents are given the turn to dispatch their schedules. The following cost based simplex method is used to implement our migration algorithm.

We consider the communication cost between all nodes to be the same. This assumption works well in a cluster environment. Our algorithm can be generalized by modifying the network cost assumption for heterogeneous networks too.

We define a cost matrix such that C_{ij} is the cost of exchanging messages between machines i and j.

Table 1: Cost matrix

Node	1	2	..	M-1	M	∑
1	C_{11}	C_{12}		..	C_{1M}	p_1
2	C_{21}	C_{22}		..	C_{2M}	p_2
..						
M-1						
M	C_{M1}	C_{M2}			C_{MM}	p_M
∑	q_1	q_2			q_M	

It is clear that

$$C_{11} = C_{22} = \ldots C_{MM} = 0$$

Communication cost for messages originating from machine *i*.

$$p_i = \sum C_{ij} \qquad (i \neq j)$$

Communication cost for messages destined to machine j

$$q_i = \sum C_{ji} \qquad (i \neq j)$$

Thus, $\sum p_i = \sum q_j$

p_i = number of agents on node *i* with cross node peers
q_i = number of agents from other nodes having their peers on node *i*.

The problem can be expressed as one of minimization:

$$Min(\sum p_i) \quad = Min(\sum q_j)$$

The requirement that the number of agents running on the hosts should be the same for load balancing is used to generate a set of constraint equations. This set of constraints is similar to the bin-packing optimization problem.

Algorithm:

1. **Find the node *i* with highest p_i**
2. **In the column of machine node *i*, find the machine *j* with the highest C_{ij} (machine with many agents having peer at machine *i*)**
3. **Compute the number of agents that can be exchanged between the two machines (*i* and *j*) without violating the load balancing constraint.**
4. **Update the table (diagonal elements C_{ii},Ca_{jj}, p_i, q_j)**
5. **Repeat steps 1-4 until no more improvement can be achieved.**

For this iteration to work, none of the migration agents should act selfishly to minimize outbound edges from its own host at the expense of the others. Because all migration agents update one another of their intentions, they iteratively build the same view of the interaction graph. They negotiate in good faith and thus, should not propose a migration scheme that undermines the other nodes.

If the final negotiated scheme results in a sizable reduction in outbound communication, and is not offset by the migration overhead, the migration will be implemented by moving outgoing agents to their respective new locations. Otherwise, the algorithm will have to be reevaluated by backtracking and removing some migration sequences, until a feasible solution is obtained.

The interval of making migration decisions is usually determined by the simulation researcher from experience on how long it takes for the interaction graph to show a steady pattern. This is a non trivial problem as it depends on the complexity of the social dynamics, the number of peers an agent may have, etc. However, one can always use a conservative estimate. In our experiments, we were able to observe that the interaction graph stabilizes mostly after running about 100 simulation steps, if the average number of peers per agent is between 10 and 20, and this estimate is likely to grow if agents have a large number of peers.

6. Experiment Design

We ran a series of experiments, on homogenous clusters and in a heterogeneous environment consisting of a cluster and remote nodes connected over the Internet. The clusters are Linux-based systems located at geographically separated sites. The first cluster has five PIII 1GHZ CPU, 1GB RAM nodes, while the second one has six P4 2.4GHZ CPU, 1GB RAM nodes. The nodes in both clusters are connected through a 1Gbps Ethernet switch. The remote nodes are a mix of Windows and Linux machines with a similar configuration connected via Internet backbone with 100Mbps bandwidth. The simulation is launched in a master-slave mode with the cluster's head-node used as the master.

The experiment on agent-interaction graph analysis and migration heuristics moreover, used a separate node that runs a JADE container connected to the main platform. This node is used as a visualization node and is not involved in the MABS execution. It collects simulation meta-data such as deployment information and inter-agent messaging data needed to visualize the interaction graph. This experiment was extended to study the efficiency of the proposed algorithm for different interaction architectures, simulation sizes and number of compute resources.

For the performance modeling experiment, a set of runtime data is collected by executing the workload on cluster and Grid-like environments. The following range of parameter values for the respective interaction architectures was used:

- Simulation size N: varied from 500 to 8000 agents, incremented in steps of 500 for each measurement instance.
- Number of compute nodes M: took values 2, 3, 4, 5 machines on a cluster, two more nodes were added for the Grid experiments. Initially, all compute nodes host equal number of agents.
- Computational granularity G: took values 1, 2, 3, 4, 5 ms.
- Outbound communication rate r: varied from 10% to 90% in steps of 10% for each measurement instance.

A set of measurement data was collected taking different combinations of the above parameters. We used the average execution time of one simulation step (p) in our performance metrics. For each input combination, we run the simulation for 20 steps. We discarded the first five measurements (to remove simulation warm up time effect) and computed the average of the remaining to get the corresponding value of p.

6.1. Experiment with peer-to-peer architecture

The peer-to-peer performance model was developed from the model of group formation simulation in social networks [9][23] as follows. It is assumed that the emergent dynamics of the simulated system leads to the formation of k clusters of agent groups where the agents in each group are expected to interact in a peer to peer fashion. The distribution of the members of a group across the compute nodes is uniform.

For a group g_i with n_i member agents $a_{i1}, a_{i2}, \ldots, a_{in}$, let n_{im} be the number of agents located on node m. Furthermore, let an agent a_{ij} in this group have interaction with p_j peer agents.

We grouped peer agents in such a way that the ratio of the number of peers hosted on the same node to that of p_j is equal to the outbound communication rate r. Agents subscribe group membership through JADE's yellow page directory service. From the yellow page entries, a simulated peer-list is

generated for each agent in such a way that a controlled outbound communication rate can be achieved.

During one simulation time, each agent sends out a message to one of the agents in its peer list and waits for a reply to update its state information and knowledge of the environment. There will therefore be *2N* messages transmitted through the system out of which *r*100 %* will be outbound.

6.2. Experiment with hierarchical architecture

To build the performance model, a case from an ongoing research project in the transportation and logistics domain was studied. The problem we considered employs a MABS approach for transportation policy analysis using a simulator tool called TAPAS (Transportation And Production Agent-based Simulator) [17]. The agent interaction follows a four-level hierarchical organization with a Transport Chain Coordinator (TCC) agent at the top, a Product Buyer (PB) and Transport Buyer (TB) agents below it. The PB agent has Production Planner (PP) agents under it, while the TB agent has Transport Planner (TP) agent below. At the lowest level, we have agents modeling factories/manufacturing sites, transporters and customers.

In this application, the higher level agents were deployed on the computer nodes based on the input parameters of the experiment in such a way as to allow control of the outbound communication. However, it is not always possible to determine the interaction pattern a-priori.

The inter-agent interaction is initiated by messages sent from customer agents to higher levels requesting for products and transportation for the products. The recipients in their turn send messages to the agents one level above them in the organization. Reply messages are then sent back from the top level agents to the bottom level in the reverse order. Higher level agents combine requests and replies from lower levels and pass to the upper ones.

The difference between peer-to-peer and hierarchical interactions from a performance point of view, is that in the peer-to-peer case the interaction is evenly distributed among all agents, while in the hierarchical case, they are concentrated towards agents at the middle layers. As a result, we expect different mathematical models to predict the performance of the two cases.

7. Results

7.1. Migration Heuristics

Figures 2a and 2b provide some information about the agent interaction graph before and after applying the reallocation algorithm. The figures help visualize the adjacency matrix of the interaction graph for 100 agents deployed on 5 machines.

Figure 2a. Pixel map of interaction graph for a peer-to-peer architecture before reallocation

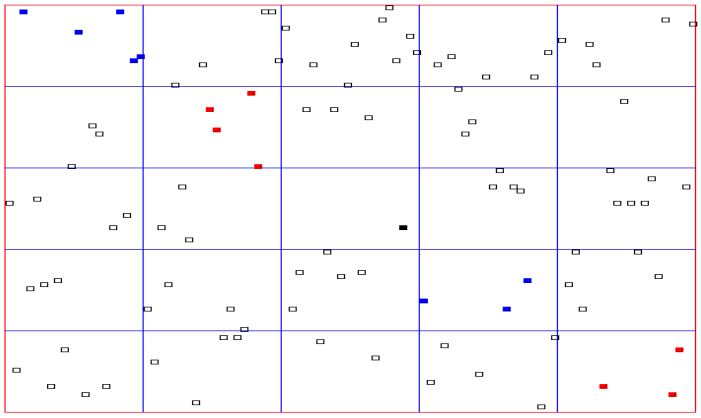
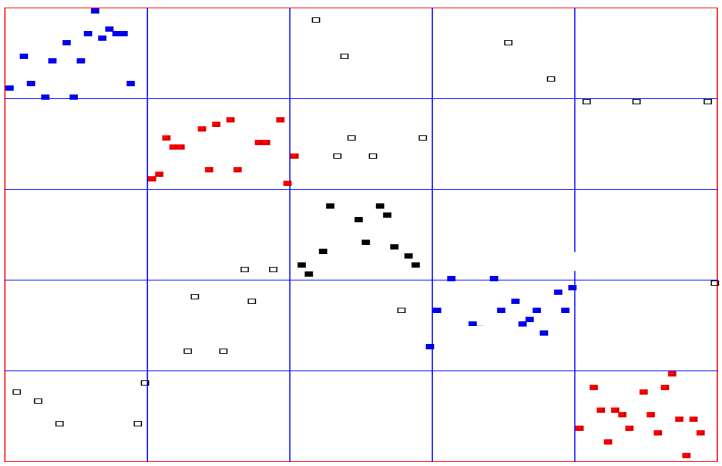


Figure 2b. Interaction graph for a peer-to-peer architecture after reallocation



The solid shaded pixels in the diagonals show inbound agent communications, while off-diagonal (transparent) pixels represent agents with outbound peers that may have to be migrated.

It can be seen that the diagonal squares are now denser, showing that, following the migration, the number of outbound links has reduced significantly. The initial outbound rate was 78% because the agents were deployed randomly and the interaction pattern emerged from the simulation over time. After the reallocation, it was possible to achieve an outbound rate of 24% by making a total of 44 migrations, and a maximum of 11 migrations from one node. The maximum is later used in a performance prediction model to estimate the cost of migration.

The peer formation is intentionally biased for the purpose of this example to form distinct and balanced clusters that results in a balanced redeployment. However, in most simulations this is rarely the case and it is difficult to achieve balanced agent clusters, or even if it is possible, it needs an unacceptably high number of migrations.

To overcome this problem, we studied the possibility of allowing imbalance on agent load distribution acrsoss the hosts for the sake of allocating as many peers on the same node as possible. It is known that when a distributed load is not balanced, the execution time is dictated by the task at the most heavily loaded host. However, the rational here is, even if the computational load is not balanced, the resulting reallocation generates less communication overhead whose gain outweighs the cost of the imbalance. The details of this work are presented in Section 7.3.

7.2. Validation of Performance Prediction Model

We built performance prediction models based on our earlier work in [5] for the workload in a cluster and Grid execution environments. With *p, N, G, r, M* given in equations (1) – (4), and estimating the performance model with a quadratic form;

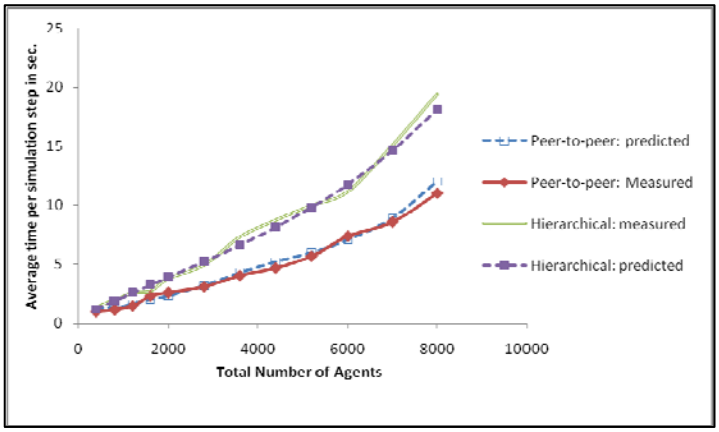
$$p = \alpha_2 N^2 + \alpha_1 N + \alpha_0 \qquad (13)$$

where the coefficients α_i *(i = 0,1,2)* are approximated from historical data of the workload execution. From our measurements, for *M = 5,* our prediction model for the peer-to-peer topology becomes:

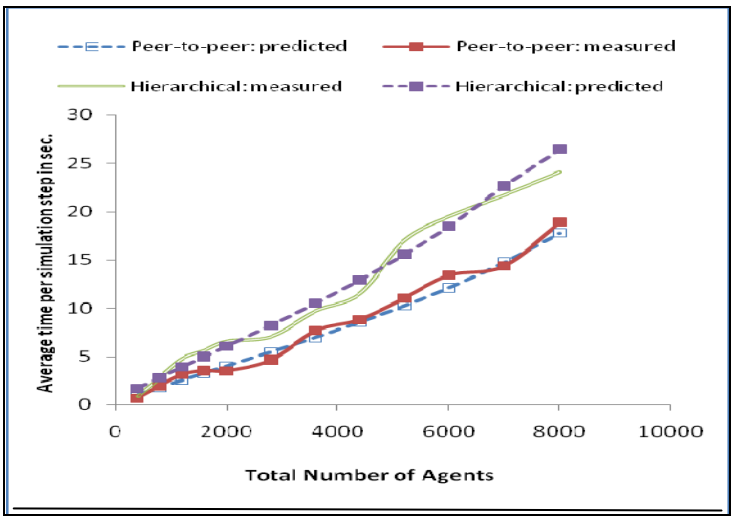
$$\alpha_2 = 0.3959\, r - 0.01224G - 0.0109r^2 + 0.0289\, G^2$$
$$\alpha_1 = 0.2613\, r - 0.1242G - 0.22616r^2 - 0.4352G^2$$
$$\alpha_0 = 0.1619r + 0.2157G - 0.05141r^2 + 0.0843G^2 \quad (14)$$

The prediction model was validated with a set of measurements and comparing the predictions with measured data and it is observed that the prediction is accurate enough, with the worst error not exceeding 11%.

Figure 3. Validation of model on a 5-node homogeneous cluster (r = 0.7, G=3)



F igure 4. Validation of model in a heterogeneous environment (r = 0.7, G=3)



Figures 3 and 4 show that larger communication overhead occurs in hierarchically organized models. Message queues will be formed at higher levels agents as they receive messages from several lower level agents simultaneously and need more time to interact with each of them.

7.3. Performance Optimization using agent migration

We assume the inter-agent interaction pattern to be a Erdos-Renyi random graph, i.e., if the agents show a fairly steady communication pattern, the corresponding entity-interaction is represented by a connected graph with fairly distinct clusters. It will then be possible to co-host agents on the basis of the clustering information so that *r* takes on a smaller value. Most of inter-agent messaging will thus be in memory operations. The implementation of this approach involves:

a. Obtaining the communication graph and generating agent clusters by analyzing the inter-agent message traffic and planning reallocation schemes as explained in Section 5;

b. Identifying the agents to be migrated through negotiation, and saving their recent states;

c. Terminating and removing the to-be-migrated agents at their current node;

d. Transferring the states to the new destination;

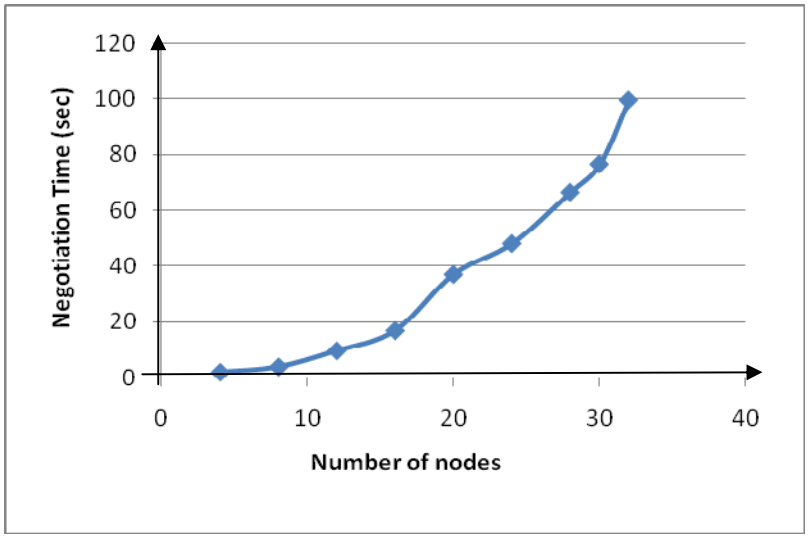
e. Launching the migrated agents at the new locations with their old state information and updating the agent directory services to reflect the outcome.

While these operations are performed, the simulation has to be suspended, and if any of the operations is not successful, it resumes from the original states. The JADE platform offers agent migration interfaces, but the performance is good enough only when the number of agents to be migrated is small [4]. Migration process as detailed above, undoubtedly entails significant overhead, and its viability should be weighed in before applying it. Unless the gain in the remaining simulation time offsets this overhead, the whole exercise will be useless.

A breakdown of the major components of migration overhead is shown in figure 6. The contributions of (*a)* above, as explained earlier is not visible as

it is entirely absorbed by the simulation idle time. The contribution of (*d)* is not significant since it essentially consists of in-memory operations and involves only a one-time state data transfer over the network.

Figure 5. Estimated negotiation time for a distributed MABS



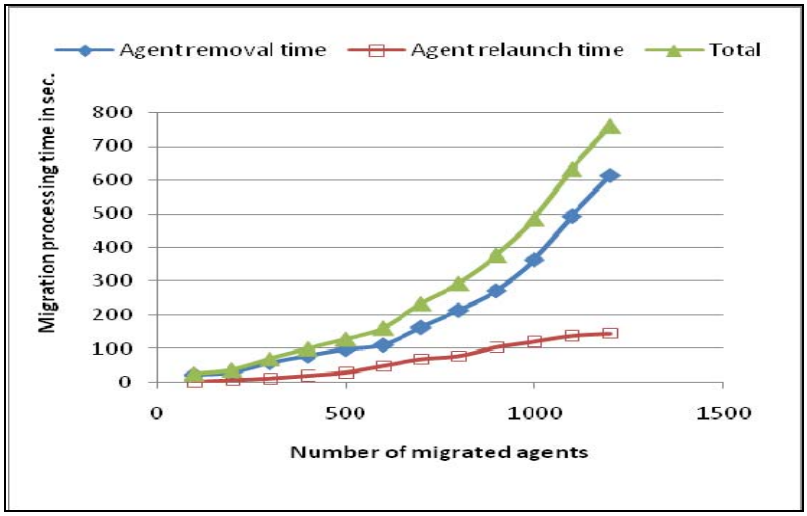
The execution time of (*b)*, the negotiation part of the algorithm, grows quadratically with the number of hosts. It therefore does not scale well if the simulation is deployed on a large number of machines. However, its overhead is still quite modest compared with what it would have been had a centralized clustering and migration approach been implemented. Figure 5 shows the time it takes to complete negotiation at one migration instance. We measured the time by executing the negotiation process on 4 to 8 nodes. For higher number of nodes, we collected the data by simulating the message exchange and network traffic on the 8 nodes.

The migration model shows the overhead incurred in moving a given number of agents from one node to another. Therefore, it is independent of any of the application parameters (like *r,* G) except the simulation size, or the number of agents to be migrated.

A prediction model of migration overhead for our experimental workload is:

$$T_{mig} = 0.0007N^2 - 0.2146N + 58.916 \qquad (15)$$

Figure 6. Breakdown of migration overhead compared to the number of agents moved



Let p_{pre} and p_{post} respectively denote the estimated execution times of a simulation step before and after migration.

If r_{pre} and r_{post} respectively represent the corresponding outbound communication rates, and the simulation runs for k_s steps, the total remaining simulation time T_{sim} with and without migration can be determined using equation (1) as:

$$T_{sim} \textit{ with migration:} \qquad T_{sim}^{\text{mig}} = ks \,.\, p_{post} \qquad (16a)$$

$$T_{sim} \textit{ without migration:} \qquad T_{sim}^{nomig} = k_s.p_{pre} \qquad (16b)$$

The predictor would recommend migration if

$$T_{sim}^{\text{mig}} < T_{sim}^{nomig}, \qquad (17)$$

For equation (17) to hold, it is required that the migration overhead is offset with the saving in the total simulation time. It then follows that migration is advantageous if the number of remaining simulation steps satisfies the following condition:

$$k_s > T_{mig} / (p_{pre} - p_{post}) \qquad (18)$$

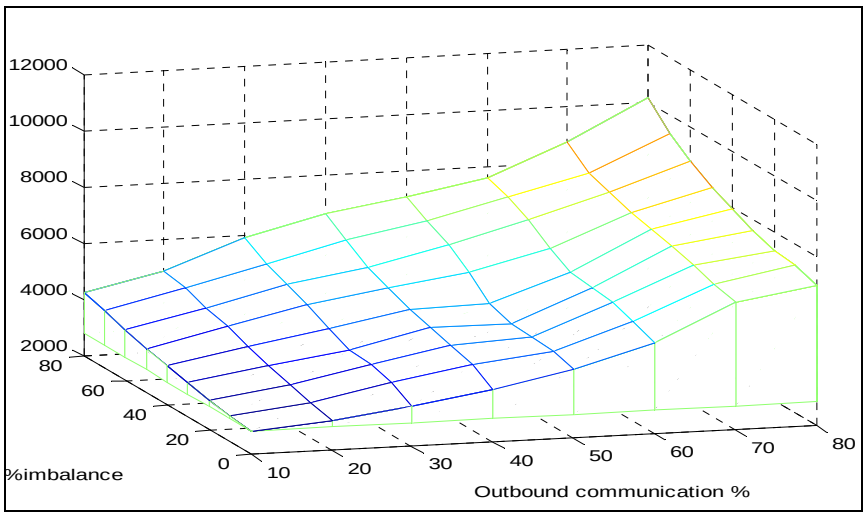
The predictor judges whether migration is a worthy action in the given situation by substituting the new value of the would-be achievable r in the performance model equations (5, 6) and estimating the remaining simulation

time. If the migration effort outweighs the performance gain, a new set of agent clusters with less migration will be iteratively computed and the model is re-evaluated until a feasible migration scheme is reached. However, since the iteration is based on heuristics, it may not always yield an optimal migration scheme.

The migration model validation experiments show that the predictions are reasonable if the number of migrated agents at a time is not too high. For example, when we tried to migrate about 300 agents from each node, the operation could not successfully complete due to network congestion and exhaustion of system resources. One way to overcome this problem over a sufficiently large remaining simulation time is to carry out the migration in two or more steps.

The performance prediction model can be extended to include load imbalance in the estimation of simulation execution time. With the help of such a model, it would be possible to decide whether a given reallocation scheme can achieve the intended gain or not. We define load imbalance as the percentage increase in the number of agents at the most heavily loaded machine compared to the average number of agents. Figure 4 shows a graph of execution time as a function of our two parameters of interest, the load imbalance and outbound communication rates.

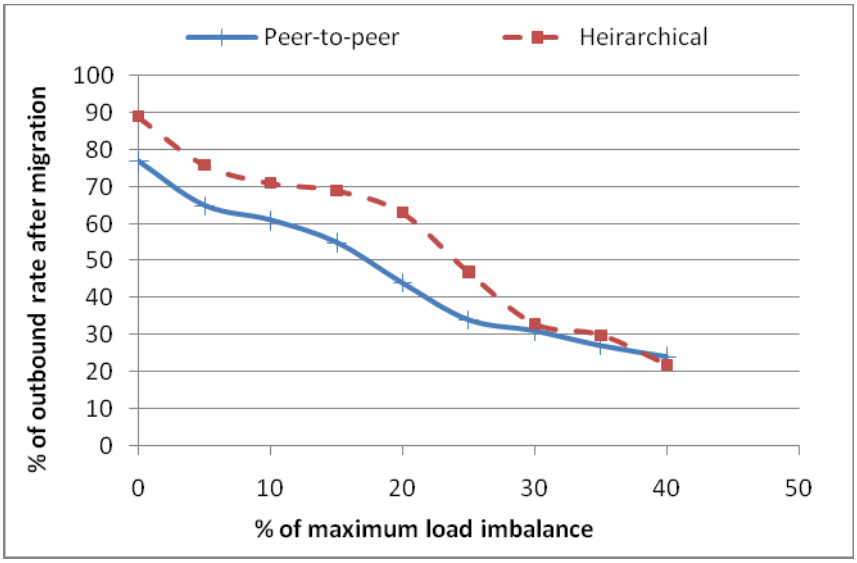
Figure 7. Execution time of an unbalanced MABS workload with 3000 agents, having a role task granularity of 1msec and deployed on 5 machines.



As can be observed from figure 7, substantial reduction in the outbound rate (from 80% to 30%) can shorten the execution time significantly even if the load has imbalance of upto 30%.

We therefore modified our migration algorithm to allow for a certain amount of load imbalance and relax the constraint equations. The modified performance prediction model to evaluate the anticipated performance gain. We tested the efficiency of the modified algorithm with permissible imbalance of 10% and 20% to compare the results as can be seen in figure 8. We simulated the effect of relaxing the constraint for a simulation size of 10,000 agents executed on 10 machines. We generated two types of workloads, a peer-to-peer and heirarchical MABS.

Figure 8. Communication overhead reduction by permitting unbalanced load at hosts.



In the peer-to-peer scenario, a static random peer formation was used. Communication clusters are gradually built around certain agents. In a practical simulation, this corresponds to the formation of groups or peers who subscribe to a yellow page service based on events of interest.

In the heirarchical scenario, we set up a communication tree of depth 4. The root agent and those immediately below it are deployed on machine 1. All other agents are delpoyed randomly on all machines regardless of their

level. The middle agents manage a lot of traffic because they communicate upwards and downwards while the leaf agents communicate with just one agent. As may be expected, the initial outbound rate for the hierarchical is worse than the peer-to-peer case. It is more difficult to regroup heirarchically organized agents if the number of high level agents is fewer than the number of machines.

8. Conclusions and Future Work

We have presented a migration algorithm that helps to reduce the communication overhead of distributed MABS applications. Unlike MABS, task migration in many distributed applications is carried out mainly for load balancing purposes by moving tasks from heavily loaded nodes to lightly loaded ones. In such applications, information on current load conditions which is obtained through relevant system calls is sufficient to make migration decisions. In the case of MABS, however, the dominant overhead comes from agent communications. Migration decisions in MABS involve a large overhead associated with the computation needed to get the pattern of the communication and to compute the agent redistribution scheme. Reducing this overhead in a meaningful way is the main contribution of this research.

The efficiency of our algorithm comes from the fact that it performs most of its computation when the application is idling due to *communication-wait* and thus does not add a considerable overhead on the overall execution time. An additional overhead comes from the use of performance models to estimate the feasibility of migration. Because the algorithm works iteratively, this overhead may be a little higher if the simulation is executed on a large number of computer nodes.

In peer-to-peer simulations, group formation and sections of interaction graphs are locally generated at the worker nodes. This removes the need for a centralized management of agent migrations and introduces self-organization features.

Because MABS have diverse applications architecture, and hence different execution behaviour, it is difficult to build a single prediction model that fits all MABS categories. However, a close look at the generic equation (14)

gives some useful clues about estimation of resource requirements for a MABS. One of the tasks we consider in our future work is to investigate the possibility of building robust performance prediction models using online performance data, without causing significant overhead.

Static agent communication patterns are assumed in this work. This assumption holds true for several use cases and can be considered as a valid starting point. In our future work, we will investigate how the algorithm can be extended to simulations where the interactions can be dynamic to some level and peer relationships among agents change.

9. References

[1] Davidsson P, et al. Applications of multi-agent based simulation, Lecture Notes in Computer Science, Multi-Agent-Based Simulation VII, Vol. 4442 pp 20-35, 2007

[2] Jang, M. and Aghaa, G. (2006) "Agent framework services to reduce agent communication overhead in large-scale agent-based simulations" Science Direct Journal of Simulation Modelling Practice and Theory Vol. 14, No. 6, Pages 679-694.

[3] Mengistu D, Davidsson P, Lundberg L (2008) Middleware Support for Performance Improvement of MABS Applications in the Grid Environment, Lecture Notes in Computer Science, Multi-Agent-Based Simulation VIII, Vol. 5003, pp 20-35.

[4] Bellifemine F, Poggi A, and Rimassa G JADE A White Paper. http://jade.tilab.com/-papers/2003/WhitePaperJADEEXP.pdf, Visited March 2008.

[5] Mengistu D. et. al. (2009) “Performance Modeling and Optimization of Agent Based Simulations in Distributed Environment” Proceedings of The Social Simulation Workshop at the International Joint Conference on Artificial Intelligence (SS@IJCAI) pages 79-92, USA.

[6] Lee J (2004) A First Course in Combinatorial Optimization. Cambridge University Press, USA

[7] Ripeanu M, Iamnitchi A, and Foster I (2001) Cactus application: performance prediction in Grid environments, Lecture Notes in Computer Science 2150, pp 807-816

8] Matousek J, Gärtner B (2007) Understanding and Using Linear Programming, Springer Berlin Heidelberg, New York,

[9] Wang F, Sun Y (2008) Self Organizing Peer-to-peer Social Networks. Journal of Computational Intelligence, Wiley InterScience, Vol. 24(3): 213-233.

[10]Chmiel K, et. al (2004) Testing the efficiency of JADE agent platform, Proc. Int'l Symposium on Parallel and Distributed Computing, Ireland, pp 49-56.

[11] Jurasovic K, Jezic G, Kusek M (2006) A performance analysis of multi-agent systems, International Transactions on Systems Science and Applications, 1(4)

[12] Vitaglione G, Quarta F, Cortese E (2002) Scalability and Performance of JADE Message Transport System, Proceedings of AAMAS Workshop on Agent Cities, Bologna, Italy.

[13] Nester C, Philippsen M, Haumacher B (1999) A more efficient RMI for Java, Proceedings of the ACM 1999 conference on Java Grande, San Francisco, pp152-159.

[14] Mengistu D, Tröger P, Lundberg L, Davidsson P (2008) Scalability in Distributed Multi-Agent Based Simulations: The JADE Case. Proceedings of FGCNS'08, China, pp 93-99

[15] Mengistu D, Tröger P (2008) Performance Optimization for Multi-agent Based Simulation in Grid Environments, 8th IEEE International Symposium on Cluster Computing and the Grid. Lyon, France.

[16] Cioffi-Revilla C (2002) Invariance and universality in social agent-based simulations. Proc. National Academy of Science USA, 99 Suppl. 3: 7314-6

[17] Davidsson P, Holmgren J, Persson J, Ramstedt L (2008) Multi Agent Based Simulation of Transport Chains, Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems.

[18] M. Low, W. Cai, and S. Zhou, "A federated agent-based crowd simulation architecture," in Proc. of 21st European Conference on Modelling and Simulation (ECMS 2007), 2007, pp. 188–194.

[19] REPAST Agent Simulation Toolkit. http://repast.sourceforge.net/ Visited January 2010.

[20] Calzarossa M, Massari L, Tessera D (2000) Workload characterization issues and methodologies. Computer Science, Performance Evaluation, Vol.1769, pp 459-482

[21] Bertelle, C. et. al. "Distribution of Agent Based Simulation with Colored Ant Algorithm". Proceedings of 14th European Simulation Symposium A. Verbraeck, W. Krug, eds. 2002

[22] Dongen S van (2000) A cluster algorithm for graphs. Technical Report INS-R0010, National Research Institute for Mathematics and Computer Science in the Netherlands, Amsterdam.

[23] Doreian P, Stokman F.N. (eds) (1997) Evolution of Social Networks. Gordon and Breach Publishers, Amsterdam, The Netherlands

[24] Xu R, Wunsch D (2005) Survey of clustering algorithms. IEEE Transactions on Neural Networks, Vol. 16(3): 645-678.

[25] The Social Media Guide. (2009) http://mashable.com/tag/data-visualization/ visited January 2010.

[26] Theodoropoulos G, et al (2006) Large scale agent-based simulation on the Grid, Proc. 6th International Symposium on Cluster Computing and the Grid, Singapore

[27] Page B, Kreutzer W (2005) The Java simulation handbook, Shaker Verlag, Aachen Germany

[28] Noack A (2007) Energy Models for Graph Clustering. Journal of Graph Algorithms and Applications vol. 11(2): 453-480.

[29] Wang F, Turner S, and Wang L (2005) Agent communication in distributed simulation, Lecture Notes in Computer Science, Multi-Agent-Based Simulation III, 3415, pp 11-24.

ABSTRACT

This research investigates approaches to improve the performance of multi-agent based simulation (MABS) applications executed in distributed computing environments. MABS is a type of micro-level simulation used to study dynamic systems consisting of interacting entities, and in some cases, the number of the simulated entities can be very large. Most of the existing publicly available MABS tools are single-threaded desktop applications that are not suited for distributed execution. For this reason, general-purpose multi-agent platforms with multi-threading support are sometimes used for deploying MABS on distributed resources. However, these platforms do not scale well for large simulations due to huge communication overheads. In this research, different strategies to deploy large scale MABS in distributed environments are explored, e.g., tuning existing multi-agent platforms, porting single-threaded MABS tools to distributed environment, and implementing a service oriented architecture (SOA) deployment model.

Although the factors affecting the performance of distributed applications are well known, the relative significance of the factors is dependent on the architecture of the application and the behaviour of the execution environment. We developed mathematical performance models to understand the influence of these factors and, to analyze the execution characteristics of MABS. These performance models are then used to formulate algorithms for resource management and application tuning decisions.

The most important performance improvement solutions achieved in this thesis include: predictive estimation of optimal resource requirements, heuristics for generation of agent reallocation to reduce communication overhead and, an optimistic synchronization algorithm to minimize time management overhead. Additional application tuning techniques such as agent directory caching and message aggregations for fine-grained simulations are also proposed. These solutions were experimentally validated in different types of distributed computing environments.

Another contribution of this research is that all improvement measures proposed in this work are implemented on the application level. It is often the case that the improvement measures should not affect the configuration of the computing and communication resources on which the application runs. Such application level optimizations are useful for application developers and users who have limited access to remote resources and lack authorization to carry out resource level optimizations.

2011:04

ISSN 1653-2090
ISBN 978-91-7295-198-3