MULTI-AGENT BASED SIMULATIONS IN THE GRID ENVIRONMENT

Dawit Mengistu

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2007:06

School of Engineering

Multi-Agent Based Simulations in the Grid Environment

Dawit Mengistu

Blekinge Institute of Technology Licentiate Dissertation Series
No 2007:06
ISSN 1650-2140
ISBN 978-91-7295-118-1

Multi-Agent Based Simulations in the Grid Environment

Dawit Mengistu

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
SWEDEN

© 2007 Dawit Mengistu
Department of Systems and Software Engineering
School of Engineering
Publisher: Blekinge Institute of Technology
Printed by Printfabriken, Karlskrona, Sweden 2007
ISBN 978-91-7295-118-1

Abstract

The computational Grid has become an important infrastructure as an execution environment for scientific applications that require large amount of computing resources. Applications which would otherwise be unmanageable or take a prohibitively longer execution time under previous computing paradigms can now be executed efficiently on the Grid within a reasonable time.

Multi-agent based simulation (MABS) is a methodology used to study and understand the dynamics of real world phenomena in domains involving interaction and/or cooperative problem solving where the participants are characterized by entities having autonomous and social behaviour. For certain domains the size of the simulation is extremely large, intractable without employing adequate computing resources such as the Grid.

Although the Grid has come with immense opportunities to resource demanding applications such as MABS, it has also brought with it a number of challenges related to performance. Performance problems may have their origins either on the side of the computing infrastructure or the application itself, or both.

This thesis aims at improving the performance of MABS applications by overcoming problems inherent to the behaviour of MABS applications. It also studies the extent to which the MABS technologies have been exploited in the field of simulation and find ways to adapt existing technologies for the Grid. It investigates performance monitoring and prediction systems in the Grid environment and their implementation for MABS application with the purpose of identifying application related performance problems and their solutions.

Our research shows that large-scale MABS applications have not been implemented despite the fact that many problem domains that cannot be studied properly with only partial simulation. We assume that this is due to the lack of appropriate tools such as MABS platforms for the Grid. Another important finding of this work is the improvement of application performance through the use of MABS specific middleware.

Acknowledgements

It is difficult to mention the names of all those without whose support I would have not overcome the challenges I met in this work. However, the following people deserve to be particularly acknowledged.

I would like to express my highest gratitude to my supervisor Professor Paul Davidsson and my examiner Professor Lars Lundberg, for initiating this research. I sincerely appreciate their guidance, insightful ideas, critical reviews and comments on my work.

The experimental part of the work would have been very difficult had it not been for the assistance of Charlie Svahnberg and Simon Kågström. The discussions I had with Kamilla Klonowska on mathematical models and performance issues were especially useful. Niklas Lavesson helped me with the final preparation of the thesis document. I would like to express my indebtedness to all of them.

Finally, I would like to thank my family for their invaluable support, encouragement, understanding and patience during my busy times.

List of Papers

The thesis is based on the following papers:

1. D. Mengistu, L. Lundberg. Survey of Grid Monitoring Tools for Domain-Specific Middleware. This paper is not published, but contains a summary of the survey conducted at the start of the thesis.

2. P. Davidsson, J. Holmgren, H. Kyhlbäck, D. Mengistu and M. Persson. Applications of Agent Based Simulation. To be published by Springer in the Lecture Notes in Computer Science series (2007)

3. D. Mengistu, P. Davidsson and L. Lundberg. Middleware Support for Performance Improvement of Multi-Agent Based Simulations in the Grid Environment. Eight International Workshop on Multi-Agent Based Simulation, May 2007. Hawaii, USA

4. D. Mengistu, L. Lundberg and P. Davidsson. Performance Prediction of Multi-Agent Based Simulations in the Grid Environment International Conference on Computer, Electrical and Systems Science and Engineering, May 2007. Vienna Austria

5. D. Mengistu, P. Davidsson and L. Lundberg. A Grid Based Simulation Environment for Mobile Distributed Applications International Conference on Multi-media and Ubiquitous Engineering April 2007, Seoul, Seoul Korea.

Contents

1. INTRODUCTION 1
   - 1.1. The computational Grid 2
   - 1.2. Multi-Agent Based Simulation (MABS) 4
   - 1.3. Motivations and research questions 7
   - 1.4. Methodology 8
   - 1.5. Contributions of the research 9
   - 1.6. Validity of results 11
   - 1.7. Conclusions 13
   - 1.8. Future work 14
   - 1.9. References 15

2. Paper I

   SURVEY OF GRID PERFORMANCE MONITORING TOOLS FOR DOMAIN-SPECIFIC MIDDLEWARE 17
   - 2.1. Introduction 18
   - 2.2. Background 19
   - 2.3. Related Work 20
   - 2.4. Survey framework 22
   - 2.5. Selected tools 24
   - 2.6. Conclusions and future work 29
   - 2.7. References 30

3. Paper II

   APPLICATIONS OF AGENT BASED SIMULATION 33
   - 3.1. Introduction 33
   - 3.2. Evaluation framework 34

3.2.1. Problem description 34
3.2.2. Modelling approach 36
3.2.3. Implementation approach 37
3.2.4. Results 38
3.3. Results 39
3.4. Analysis 42
3.4.1. Problem description 42
3.4.2. Modelling approach 43
3.4.3. Implementation approach 44
3.4.4. Results 45
3.4.5. Limitations of the study 45
3.5. Conclusions 46
3.6. References 48

4. Paper III
MIDDLEWARE SUPPORT FOR PERFORMANCE IMPROVEMENT OF MABS APPLICATIONS IN THE GRID ENVIRONMENT 51
4.1. Introduction 52
4.2. Motivations for the study 53
4.3. Performance of MABS applications on the Grid 54
4.3.1. Key features of MABS affecting performance 54
4.3.2. Performance analysis of Grid applications 55
4.4. Proposed architecture 56
4.5. Experiment 59
4.5.1. Workload characterization 59
4.5.2. Grid environment 61
4.5.3. Implementation of workload 62
4.6. Results 63
4.6.1. Effect of multi-threading on performance 64

4.6.2. Effect of outbound communication 65
4.6.3. Effect of time-step synchronization 65
4.7. Middleware support for performance improvement 66
4.8. Conclusions and future work 68
4.9. References 69

5. Paper IV
PERFORMANCE PREDICTION OF MULTI-AGENT BASED SIMULATION APPLICATIONS ON THE GRID 71
5.1. Introduction 71
5.2. Multi-agent based simulation on the Grid 73
5.2.1. Multi-agent based simulation 73
5.2.2. MABS features 73
5.2.3. The Grid 74
5.3. Methodology 75
5.3.1. Workload modeling 75
5.3.2. Performance metrics and measurement 76
5.3.3. Performance modelling 76
5.3.4. Implementation 78
5.4. Results 79
5.4.1. Performance modelling with synthetic workload 79
5.4.2. scalability 81
5.4.3. Prediction with variable granularity 83
5.4.4. System utilization 84
5.5. Discussion 85
5.6. Conclusions and future work 86
5.7. References 86

6. Paper V

A GRID BASED SIMULATION ENVIRONMENT FOR MOBILE DISTRIBUTED APPLICATIONS 89

6.1. Introduction 89

6.2. Background and motivations 91

6.3. Related work 92

6.4. Multi-agent modelling 93

6.5. The simulation environment 94

6.6. Experiment and results 96

6.7. Conclusions and future work 102

6.8. References 103

ONE

I. INTRODUCTION

The emergence of the computational Grid brought new opportunities to solve grand problems in science. One application area which can benefit from the Grid is the field of simulation, in particular multi-agent based simulation (MABS). Executing large-scale MABS applications requires huge computing resources that may not be available without the Grid. However, due to heterogeneity of Grid resources, the unpredictability of the execution environment, and the execution characteristics of the application itself, these applications may not always run efficiently on the Grid. The objective of this thesis is to study the performance characteristics of MABS applications in the Grid environment, and to propose a domain-specific middleware support for performance improvement.

Understanding the execution behaviour of an application is a crucial step to design and develop efficiently running systems on the Grid. This behaviour depends a lot on the functional requirements of the application and its architecture, and needs to be investigated thoroughly. Since the Grid is a dynamic execution infrastructure, applications designed for deployment on the Grid also need to be dynamic and adaptable to the changes in the execution environment and be able to maintain a desired level of performance. In the Grid environment, application adaptivity is usually achieved through support from a middleware. This middleware monitors the run-time behaviour of the application and the infrastructure it is running on, and processes the collected data to optimize the execution of the application in response to changes in the environment. Since the characteristics of potential Grid applications are diverse, it may not be possible to develop a general middleware suitable for applications. It is therefore necessary to develop domain specific middleware to support optimal performance of the applications.

In this chapter, a background of the Grid as a computational environment, and a brief overview of MABS is given in Sections 1.1 and 1.2 respectively. The motivations for this research will then follow in Section 1.3. The methodology followed in addressing the research questions is presented in Section 1.4. Section 1.5 discusses the contributions of this research. The validity of the findings of this work will be covered in Section 1.6. We then present our conclusions in Section 1.7, followed by future work in Section 1.8.

1.1 The Computational Grid

The availability of faster processors, high speed networks and storage devices, and the maturity of distributed software services gave way to the vision of a wide-area distributed computational framework. The Grid is a realization of this vision and is defined in different ways based on the interpretation of this vision. To mention two of these definitions, according to Foster and Kesselman, the computational Grid is:

"A hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [1]

The Global Grid Forum (www.ggf.org) defines the Grid as:

"A system that is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports collections of users and resources (virtual organizations) across traditional administrative and organizational domains (real organizations)."[2]

The Grid is thus a technology that provides users with uniform access to distributed resources and services through a framework of layers of software services. The software layers or middleware provide services for application scheduling, execution, resource management, security, data access, etc. As can be seen from the definitions above, the Grid may consist of heterogeneous resources. However, the middleware services should hide this heterogeneity from the users and provide seamless access, creating the illusion of a single powerful virtual computer.

Because of the heterogeneity of Grid resources and its capacity to serve a large number of users concurrently, the demands for computational power fluctuate and application execution is generally unpredictable. It is therefore required that the Grid execution and resource management middleware services respond to load fluctuations in due time. The middleware consists of monitoring and information services that consistently observe the operation of compute and data resources, network traffic and the execution of jobs. The monitoring data is extremely useful to make scheduling decisions, suspension, termination or migration of applications with poor performance, as necessary.

Although the Grid monitoring middleware provides an indispensable support to make accurate scheduling decisions and achieve the desired quality of service, they are mostly effective in remedying infrastructure related

problems only. Assignment of tasks to compute resources that do not match the task's resource requirement could reduce execution performance or be a reason for underutilization of resources. Grid applications submit their resource requirements and performance models to the Grid resource broker. The Grid resource broker then finds suitable resources for these applications by querying the Grid monitoring and information services. The resource requirements include machine architectures, operating system requirements, processor speed, memory, execution libraries, tools etc [3].

The resource requirement metadata of applications is usually obtained in one of the following ways [16]:

- Theoretical prediction: making a prediction based on analysis of an application's programming model and problem domain;
- History based prediction: making a prediction based on previous runs;
- Test-cases based prediction: the application itself is tested for a limited set of cases on representative machines.

If performance models of the application are available, it will be possible to [3]:

- Evaluate time predictions for different resource allocation strategies;
- Explore the scalability of the application and the resources;
- Predict system resource usage.

As the size and complexity of an application grows, it may be necessary to partition the application into sub tasks that are deployed on several Grid resources. If an application is run on a set of homogeneous resources, all sub tasks should have balanced computational sizes. However, in reality, Grid resources are heterogeneous and such kind of partitioning results in poor performance because the over all execution time is dictated by the slowest resource. It is therefore required to apply load balancing techniques to ensure fair distribution of tasks on the Grid resources considering their capability and current load conditions.

If a Grid application is divided into sub tasks that ought to communicate and synchronize with each other to share data and states, load balancing will become a complex undertaking. Even if load balancing is carried out using a generic strategy or by employing application-specific heuristics, the initial allocation may not be useful throughout the execution of the application, as the Grid environment changes through time. Network traffic, resource failure, load fluctuation, etc., can easily render the initial allocation ineffective. It will be necessary to draw a new scheme of task allocation and

migration of tasks as appropriate. The Grid resource management service on its own will not be able to fully handle such types of dynamic load balancing requirements and needs to be assisted by application specific middleware support tools.

The main objective of this research is to improve performance through middleware support.

1.2. Multi-Agent Based Simulation (MABS)

MABS is a modern simulation technique used to study complex systems and phenomena characterized by entities having social and autonomous behaviour. At present, MABS is replacing or complementing traditional equation-based macro-simulation techniques in scientific domains such as economics, sociology, biology, ecology, etc.

MABS is a discrete event simulation (DES) technique that can be used for several purposes such as studying the impact of certain effects in organizational settings. It gives insight into the dynamics of social phenomena and project scenarios by formulating and testing models of existing phenomena through computational reconstruction [6]. 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." [7]

The individual beings constituting the real world phenomena to be simulated (such as human beings) are modelled by agents capable of pursuing their objectives, making decisions autonomously and communicating with other agents to create social structures. An agent is an autonomous computational entity that exists in the form of programs, having sensory and perceptual capabilities, and can make decisions and act on the environment it is embodied in through effectors [6]. The sensors communicate with the environment to receive information needed for the agent to update its knowledge of the environment. The effectors also communicate with the environment to convey the actions the agent takes to change the state of the environment. An agent thus has knowledge of its environment, mental states

and beliefs it regularly updates using the information gathered through its sensory components.

Agents can also communicate with each other to influence one another or to pursue collective intentions. For agents to understand each other, the communications should take place using standard language constructs. The Agent Communication Language (ACL) proposed by the Foundation of Intelligent Physical Agents (FIPA) is the commonly used language based on the speech act theory [8].

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.* [9]

At the computational level, the program codes of agents representing the simulated entities are realized as executable threads. A MABS application is therefore a parallel application that is executed on parallel architecture computers. The interaction between the real world entities is modelled by inter-agent communication, and at the computational level, it is realized by inter-thread messaging or information exchange through a shared memory. From computational point of view, the complexity of a simulation may be determined by:

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

An important aspect of parallel simulation is time management, preserving the temporal characteristics of the simulated phenomena. The ordering and causality of events should be properly captured to reproduce the behaviour of the real world problem and to ensure the repeatability of the simulation experiment. Enforcing causality of a MABS application is a challenging task when simultaneous real world events are to be simulated using a multi-threaded application on a single processor. In effect, multi-threading in this case is only an illusion, introduced to mimic concurrence of events. Because the simulation code is executed sequentially, timing of

events should be handled with care, or the causality of events will be violated. The time management aspect of discrete simulations is distinguished by the mechanism used in the application to advance simulation time [10]:

- Time-stepped 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: 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.

MABS application developers should therefore include appropriate time management features into their applications carefully.

In most cases, developing the applications from scratch is a time-consuming task and prone to faults. To overcome this problem, MABS platforms capable of handling the most generic features of MABS are developed. The simulation platforms are important methodological tools that aid researchers in the process of modelling and development of simulation applications [14]. They 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 platform involves:

1. Setting up and initialization of the MABS platform in a hosting environment;
2. Creating instances of simulation agents with the platform supplied templates;
3. Implementing additional features of the agents and the communication behaviour by writing program codes as necessary;
4. Launching the simulation application.

An interesting challenge with carrying out certain social simulations is the requirement that the entire phenomena should be modelled and implemented in the simulation application because the findings of partial simulations cannot always be generalizable and do not produce meaningful results. (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." [9].

Consequently, a MABS intended to simulate a phenomenon having a huge number of individuals is basically a large-scale application and requires huge computing resources that is not commonly available on a single machine. Large-scale MABS can thus be realized as a parallel distributed simulation. (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."

The computational Grid can provide the required resources to run large-scale MABS as a distributed parallel application. In addition, the Grid has several important features that make it suited for MABS, such as service factories and life-cycle management, which allow distributed agents to be instantiated and have a controllable level of persistence [11].

1.3. Motivations and Research Questions

Although the Grid can offer the much needed computational infrastructure for large-scale MABS applications, it is difficult to effectively utilize its reach pool of resources due to the inherent features of MABS. Inter-agent communication that takes place between agents running on geographically distributed resources is a source of latency and performance degradation. The time management aspect and the need to maintain global time synchronization also affect application performance severely. Thus, the simulations may overload the Grid resources by creating a large amount of network traffic and consume a lot of processing time and storage without making use of it. The deployment of MABS applications on the Grid therefore requires understanding these architectural features and finding effective solutions.

The focus of the research is the ***efficient execution of large-scale MABS applications on the computational Grid with the support of a domain-specific middleware***. The task of this middleware is to study the application execution environment and provide a mechanism to achieve better performance. Information about the current status of the execution

environment is obtained through Grid monitoring services to predict the performance of the application. It is desired to have a good knowledge of the Grid execution environment, the nature of the simulated problem in general, and its specific behaviour to propose an effective middleware solution. The key issues around this research can thus be summarized by the following research questions:

RQ1: *What are the currently available performance monitoring systems that can be used in conjunction with the envisaged middleware?*

RQ2: *Which are the MABS environments commonly used in large-scale MABS? How widely are they deployed on the Grid?*

RQ3: *What are the key features of MABS that significantöy affect application performance in the Grid environment?*

RQ4: *How can the effects of these MABS features on application performance be quantified?*

RQ5: *What specific techniques would an application support middleware employ to improve the performance of MABS applications?*

1.4. Methodology

Survey and experimental methods were used to deal with these issues, specific cases depending on the nature of the respective sub question.

To address RQ1, it is necessary to analyze the available Grid performance monitoring tools, to see if they could fit into our context. An extensive literature survey was conducted to gather information on popular Grid infrastructure and application monitoring tools. Because there are a large number of such tools, it was necessary to prepare a selection and evaluation framework to limit our study to those tools which comply with current Grid standards and provide real-time instrumentation. We further narrowed down our list of tools to those which include API support for different programming languages, and are easily available for download and deployment in an experimental environment.

RQ2 is also tackled through a survey work. We collected publications describing actual implementations or implementation frameworks for anticipated MABS applications. We prepared a comprehensive evaluation framework to categorize MABS applications.

An experiment was designed to deal with the issues raised in RQ3. The prominent features of MABS that influence the execution behaviour of an application are studied in the experiment. A Grid testbed of Linux machines running Globus 4 Toolkit was setup for this work. A synthetic MABS workload was developed as a set of Java web services that are deployed on the Grid nodes which host the multi-agent execution environment. There is also a client side component responsible for the coordination of the services and performance instrumentation purposes.

We followed an evaluation approach to measure and compare the effects of the studied features. It was found that the scalability and performance of a MABS application on the Grid is affected by the size of the simulation, the granularity, the number of Grid nodes employed and the amount of cross node inter-agent communications.

To answer RQ4, the experimental system was extended to allow a systematic measurement of cause-effect relationships. The *factorial design* method was followed since there are four independent variables affecting application execution characteristics. After taking a large set of measurement data from the experiment, an empirical performance model was developed for the application.

Several approaches can be used and solutions be proposed to deal with RQ5. One solution is the reduction of waiting time by keeping the number of inter-node messages to a minimum. Another approach is to employ message aggregation techniques to reduce communication overheads. These techniques can be implemented as part of the middleware layered between the simulation application and the Grid resources. Experiments were conducted to evaluate the effectiveness of these approaches.

1.5. Contributions of the Research

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

Chapter 2 (Paper I)

The work covered in this chapter addresses RQ1. In high performance computing systems, it is generally difficult to achieve a meaningful improvement in performance without employing appropriate monitoring tools. The need for monitoring tools is more critical in heterogeneous

environments such as the Grid where analytical models of the computing environment or the application alone do not yield the expected performance results. A comparative evaluation of common Grid infrastructure and application monitoring tools is presented in this chapter. An evaluation framework was established based on recommended Grid monitoring architectures (GMA). The results discussed in this paper are useful to middleware and application developers in general.

Chapter 3 (Paper II)

This chapter presents a survey of MABS applications and addresses RQ2. The findings of the survey are comprehensive in that they deal with a wide range of issues beyond this research question. They show that MABS are used in simulating social, organizational, economic, biological, etc., systems. It is also observed that in some domains, the simulation requires the deployment of a large number of agents. However, partial simulations were carried out to study the problem domain in those cases where full-scale simulations were needed, possibly due to lack of computing resources (hardware and software). Another observation made through this study is the lack of simulation environments or MABS platforms. Because most social simulation applications are developed from scratch, we could not find representative platforms to be extended to the Grid environment. The proliferation of large-scale MABS in the industry is still at its infancy, and more work is yet to be done to exploit the full benefits of MABS.

Chapter 4 (Paper III)

The work presented in this chapter has two contributions leading towards the anticipated research goals.

First, it investigates the issues raised in RQ3. It presents a detailed account of the performance characteristics of MABS applications in the Grid environment by identifying the architectural features of MABS that affect performance. The extent to which these features degrade performance is examined. It is found that massive multi-threading, inter-agent communication, time-step synchronization and the granularity of the agents' computational task affect the performance of a MABS application severely.

The paper also partially provides answers to RQ5. It covers a preliminary experiment to develop a middleware that contributes to performance improvement by reducing communication that takes between agents running on different Grid nodes. It is also shown in this experiment

that it is possible to reduce execution time through message aggregation techniques where communication operations are initiated at the end of a simulation cycle. Messages are transported to and from a node in bulk rather at the instance of creation, to reduce communication overheads significantly. The findings of this work are useful for MABS platform and application developers, as they reveal the factors one should pay most attention to in designing the architecture of a MABS.

Chapter 5 (Paper IV)

This chapter addresses RQ4 by presenting a performance prediction model for a synthetic MABS application and shows that the qualitative findings of the experiment can be extended to other applications as well. The main contribution of this chapter is the development of a prediction model that can be used for scheduling and load balancing activities in MABS applications. One challenge with building performance models for MABS applications is that, unlike object oriented applications, the execution behaviour of agents is generally unpredictable as the course of action an agent takes and the amount of computation it performs cannot be known in advance. In this work, an experiment was designed where the agents perform varying amount of tasks bound within a predetermined range. It was found that the prediction models can be applicable even with an application whose granularity has a high level of fluctuation.

Chapter 6 (Paper V)

This chapter studies the issues raised in RQ2. It presents a Grid-based simulation environment for mobile distributed applications. It is part of an ongoing work to exploit the huge computational resources offered by the Grid for solving real world problems. The experiment used in this work was aimed at solving traffic congestion problems using the multi-agent approach. The limitations of current works and existing tools, the feasibility of using the MABS approach, and the need to build a simulation environment on the Grid are discussed in this paper. Preliminary results obtained using a prototype of the proposed simulation environment were also discussed.

1.6. Validity

Although the results obtained in the experiments are encouraging, there are also limitations that need to be acknowledged. We group these limitations

or threats to validity of the results into two: external and internal [12]. Both types are discussed below:

External Threats (Generalizability)

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

Application: In the 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. Some of the prominent features of a MABS application used in modeling social simulations such as spatial explicitness, mobility, etc., were not included in our workload. A further simplification in the workload was that it does not contain any I/O related activities. In reality, few simulations of the envisaged type are like that.

Infrastructure: although it was attempted to make the compute resources heterogeneous by exposing them to different load conditions, the study would have been more comprehensive had heterogeneity in hardware and execution environment been included. Because the Grid nodes are connected in one network, the effects of communication problems experienced in a production Grid were not addressed. Consequently, the effect of network latency was also not accounted for in our experiments.

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.

Instrumentation: Time measurements were not precise enough since the accuracy of the API used in the application is in milliseconds. It was not possible to precisely measure the duration of many important events. As a result, we had to take averages of the measured values by running events of interest in several loops, multiple times. The time-granularity of the applications is also not accurate enough to give a perfect prediction model. Another area where measurement uncertainty is encountered is in the measurement of CPU utilization.

Measurement data: The data produced in this work is not sufficient to make a conclusive argument about the phenomenon under consideration. The

factorial design approach we used to facilitate a systematic measurement process, though important, did not include a wide range of input values. It can therefore be said that the model is valid only over a limited range of values and should be extended with a much larger data set.

Analysis: The mathematical methods used in the prediction scheme employ linear approximation techniques and are not precise enough. We used the generalized least squares (GLS) method to build the performance prediction model. This model assumes a linear relationship between cause and effect, at least over what is commonly called a range of operation, a set of input values of interest to the modeller. This limitation, combined with the uncertainties in the measurement process make the generality of the established performance prediction model questionable.

1.7. Conclusions

The principal objective of this thesis and its contribution revolve around performance issues of MABS applications. As MABS is a relatively recent comer to the distributed computing area, it was not possible to find sufficient literature and deployed systems that could be used as stepping stones for our work. Therefore, most of the work reported in the thesis can be described as a preliminary research.

The MABS specific issues that adversely affect execution efficiency in the Grid have been investigated. An important finding is that although several factors are apparently recognized as the most severe causes of performance degradation, it is observed that the influence of some of these factors is dominated by the others to the extent that they may not be sources of concern at all. For example, it was seen from the experiments that with increase in the level of multi-threading (simulation size), the effect of network latency is eclipsed by that of context switching and thread scheduling.

The challenges and difficulties to predict the behaviour of MABS applications and the need to deal with it in a different way from traditional approaches is well observed in this research.

Although the findings reported so far constitute a necessary step towards the eventual achievement of our objectives, further research is necessary to investigate the remaining issues. We identify some important issues to be addressed in the next section.

1.8. Future Work

The work accomplished so far covers a necessary phase of large-scale MABS application development on the Grid. There is more work to be done to improve the validity of the obtained results, to provide more comprehensive performance improvement solutions, and to address other remaining issues. A list of areas of improvement and extensions we consider necessary are presented below.

As explained earlier, the mathematical models developed for performance prediction need to be improved. A wider set of input data and measurement should be used to obtain a more accurate prediction model. In addition, it is necessary to conduct the research in a Grid environment with some level of heterogeneity, fluctuations in communication and load characteristics. We therefore plan to perform future studies in a more realistic setting.

Although the synthetic workload we used in our previous experiments helped us gain better understanding of performance issues, a more comprehensive study should be developed based on real world MABS application. We will use some of the existing MABS applications as real workloads in the upcoming study. We shall take representative MABS applications and investigate their behaviour to determine whether generic prediction models and middleware support would suffice to improve performance, or if it is necessary to establish domain specific components on a case by case basis.

On the middleware side, there is a requirement to incorporate more components for performance improvement. Although some of these components are generic and can be directly imported from existing products, it is necessary to add certain extensions to them. We recognize the following requirements as key to efficient execution of MABS and should be included in the middleware:

1. Dynamic load balancing in a heterogeneous Grid environment. Because applications run in a dynamic environment where the compute nodes have different load levels, there is a requirement for dynamic performance prediction, load balancing and scheduling. The middleware should sense load fluctuations and changes in the infrastructure and make on-the-fly decisions on load balancing and migration of tasks as necessary.
2. Improvement in local scheduling decisions. Since a MABS application is massively multi-threaded, deployment of such an application can strain

the fair utilization of processor time at compute nodes shared by other loads. The scheduling policies of some local operating systems may only identify processes at thread level and attempt to allocate the same size of time slice to all user threads, irrespective of which process they belong to. Hence, there is a possibility that a massively multi-threaded MABS application can unfairly take a larger share of the processor time stalling other concurrently running processes. Such problems are frequently encountered at Grid nodes which volunteer to donate their free time only. Since the incident is either not visible to the local operating system, or the Grid user has no control over them, a remedy can only be found at the application level. The middleware should therefore configure the application at run time, to adjust the synchronization and time management components, to pre-empt potential unfair exploitation of resources by MABS applications.

1.9. References

[1] Stockinger, H., "*Defining the Grid: a snapshot on the current view*." Springer Science+Business Media. 2007

[2] The Global Grid Forum (www.ggf.org)

[3] Bailey, D.H., and Snavely, A. "*Performance modelling: Understanding the present and predicting the future*." EuroPar 2005. Lisbon Porutagal. 2005.

[4] 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

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

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

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

[8] Bradshaw, J.M. "*An Introduction to Software agents*" MIT Press, Cambridge USA. 1997. pp. 346.

[9] 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.

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

[11] 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. V 3415, 2005. pp. 1-10.

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

[13] Theodoropoulis, G., Zhang, Y., Chen, D., and Minson, R. "*Large-scale distributed simulation on the Grid.*" Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops (CCGRIDW'06). Singapore. 2006.

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

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

[16] Nabrzyski, J., Schopf, J. and Glarz, J. "*Grid Resource Management: State of the art and Future Trends.*" Klauwer Academic Publishers, Norwell, MA, USA. 2004.

TWO

Survey of Grid Monitoring Tools for Domain-Specific Middleware

Dawit Mengistu, Lars Lundberg
Department of Systems & Software Engineering
Blekinge Institute of Technology, Ronneby, Sweden
E-mail: {dawit.mengistu, lars.lundberg }@bth.se

Abstract

Resource management systems are essential for the efficient operation of the Grid execution environment. However, these systems do not always achieve the expected performance level for certain classes of application domains. The major cause of performance degradation here is often related to the inherent behaviour of the applications themselves. A domain-specific, performance aware middleware used in conjunction with the Grid resource management systems can help overcome this problem. The effectiveness of such middleware heavily relies on Grid monitoring tools which provide application, compute and network performance information for performance modeling and prediction purposes. It is therefore necessary to study existing Grid monitoring tools and evaluate their capabilities to identify those suitable for such middleware. A survey of Grid performance monitoring tools and an overview of a performance aware Grid middleware for multi-agent based simulation applications is presented in this paper.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Performance – *Design studies, modeling techniques, measurement techniques.*

General Terms
Performance, Experimentation.

Keywords
Application Performance, Grid Middleware, Performance Monitoring Tools, Grid Computing.

1. Introduction

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 bottleneck [1].

The implementation of performance analysis techniques in the context of Grid computing is more complex than in other paradigms, due to the nature of Grid resources, such as heterogeneity, availability, load characteristics, etc. Traditional performance measurement tools for parallel and distributed systems fall short of addressing the basic performance requirements of the Grid. It is, therefore, necessary to look for other tools. The benefits of such tools are clear to all stakeholders in Grid computing, particularly to Grid application and middleware developers.

Grid middleware generally facilitate execution of applications with fair utilization of Grid resources. Applications with statically defined execution characteristics are less likely to execute efficiently in all Grid environments even if the jobs are managed by very good schedulers. Due to the heterogeneity of resource and the dynamic nature of the Grid, relying on the resource matching capabilities of Grid schedulers alone does not guarantee execution optimality. This problem can be overcome by implementing a domain-specific, performance aware middle-ware to adapt applications to the dynamic environment in which they are executed. We plan to use this middleware as a layer between our application of interest, multi-agent based simulation (MABS) applications and the Grid. It optimizes a MABS application run-time code for efficient execution in the Grid environment. Performance monitoring tools support the operation of such middleware by providing relevant information for resource and application modeling, performance prediction, application optimization, and job submission. It would therefore be necessary to identify appropriate Grid monitoring tools that can work in conjunction with the respective middleware.

In this paper, we present a survey of relevant performance monitoring tools used in the Grid environment. The rest of the paper is organized as follows. Background information on the domain of interest, multi agent based simulation, is presented in the next section. Multi agent frameworks and

issues relevant to application development are also discussed in brief. Section 3 explains how the monitoring tools presented in this paper are selected. It also explains the features desired of such tools to meet the requirements of middleware development. A brief introduction to the essential features of the selected tools and a summary of the observations made about the tools in this work is given in Section 4. We then conclude this paper by summarizing our findings and citing directions of future work.

2. Background

Multi Agent Based Simulation (MABS) is replacing traditional equation based modeling and simulation techniques in the study of complex systems in different real world phenomena. Agent-based modeling and simulation is appropriate for domains characterized by a high degree of localization, distribution, socialization of entities, and discrete decisions. Few examples of application domains include: telecommunications and transportation traffic, financial markets, and coordination of autonomous mobile robots. A multi agent system (MAS) is a distributed system consisting of software agents loosely coupled to work together to solve problems that are beyond their individual capabilities or knowledge of each entity [18]. An agent represents an entity in the real world and its operations in the multi agent system include: gathering information (sensing), analyzing the information and making decisions for next course of actions, and take actions accordingly.

Although a multi agent system is characterized by its distributed behaviour and relative independence among agents, a core requirement for the agents is that they achieve their objectives through interaction, negotiation and cooperation. Communications between agents usually take place through standard protocols. The Agent Communication Language (ACL) is commonly used for communication between agents [21].

Subject to the number of agents and the complexity of their functions, developing the simulation application may be time consuming and difficult. The use of generic platforms is more popular as building agent based applications from scratch suffers lack of consistency and uniformity. A MABS platform provides the physical structure in which agents can be deployed. It consists of agent management system, a directory service, a message transport service and the agents themselves [19]. Using MABS platforms, users can model their simulation problems with relative ease. Several platforms are available to date, with some of them focused on specific application domains. A few of these platforms allow the user to write

additional codes for managing specific tasks, creation and removal of agent instances, control of communications, etc.

Technically, a MABS application is a multi agent system comprising several agents working in tandem performing computing, communication and data access tasks. It is therefore natural to develop a MABS application as a parallel application. The Grid is good for MABS applications because simulations involving thousands or millions of agents and highly complex and data intensive tasks can benefit from the Grid's high throughput resources.

Static and performance insensitive MABS applications cannot be good workloads for the Grid. It is therefore necessary to have a MABS platform supported by a middleware that can 'sense' the execution environment, the behaviour of the application and adapt it for best performance. To achieve this, a performance aware middleware should be integrated with the MABS environment. With the support of the middleware, it is possible to tune the simulation application for optimal execution in the Grid.

Grid applications are categorized by features characterizing the programming models to be employed. They can be based on RPC, message passing, Java RMI, or task parallelism [20]. The implementation details of these categories have differences and require specific methodologies for development and execution of the applications. It is imperative to identify an appropriate programming model that enables MABS platform and application development for Grid, employing the envisaged performance support middleware effectively.

The architecture of the MABS application itself has also an impact on the effectiveness of the methodology applied in its implementation. The behaviour of the agents appearing in the simulation such as size of computational code (weight), mobility, autonomy, mode of message transport, type of directory/information service, etc., influence how efficiently the application can be executed in a particular Grid execution environment. Parametric sweep applications are usually easier to write for the Grid. They are suited for studying some simulation models including MABS. Job specifications for such applications can be flexibly modified during run time to match performance predictions for load balancing and scheduling.

3. Related Work

Several studies have been conducted towards the development and implementation of robust Grid applications. In the following discussion, only

three approaches are presented to provide information on ongoing works. The presentation given is very brief, just to show the work being done in this area.

Grads: The Grid Application Development Software (GrADS) project is initiated to solve performance optimization problems. The other objective of project is to minimize the coding effort in Grid application development through the use of standard libraries [20]. A program development and execution structure called GradSoft is developed for this purpose. Performance responsiveness is implemented in GradSoft in two ways: modifying the application source using performance monitoring feedback and recompiling it, and/or by adapting the compiled application to the run time environment through dynamic binding and resource matching. These processes are realized by cascaded components from code generation to application execution.

PACE: The Performance Analysis and Characterization Environment (PACE) toolkit was originally developed to support performance improvement in parallel and distributed applications. It monitors the application and resources separately and generates application and resource models to be used by an evaluation engine to predict performance, and eventually support scheduling. The models can be used for performance optimization of applications and to study the scaling effect of increasing the number of processors or the impact of modifying a processor mapping strategy [21].

Auto Pilot: This toolkit is designed for real-time application and resource monitoring and control built on the Globus Toolkit. It consists of three major components: distributed sensors for performance data acquisition, actuators for implementing performance optimization decisions, and a decision making infrastructure for evaluating sensor inputs and controlling actuator inputs [20]. Auto Pilot allows inserting instrumentation codes into an application's binary. For the decision making part, the toolkit uses artificial intelligence techniques such as fuzzy logic. This toolkit uses control theory concepts in all its components: sensing, decision making and actuation.

Although the above approaches attempt to address the problem and provide solutions as explained earlier, they may not always guarantee performance optimality for all applications due to their generic features. If one wishes to accommodate specific needs of different application domains in these tools, it is necessary to extend their features by adding additional components to them. This will, however, increase the computational and communication overhead associated with the added components. It is therefore preferred to have domain-specific tools particularly tailored to a certain class of applications.

Grid application development has basic similarities with the well known parallel and distributed computing paradigms. System management issues such as application sub tasking, separation of the application into master (client) and worker (resource) side codes, communication of partial results between tasks and Grid nodes, etc., are dealt with in a more or less standard manner. Performance monitoring tools are employed to measure the level of computation and communication, the amount of coupling between different parts of the codes, the utilization of resources, the latency of the network, etc.

4. Survey Framework

A directory of available performance monitoring and analysis tools for Grid is available in a white paper [2]. A survey made on performance analysis and visualization tools for parallel programs in multi processor systems takes account of the reporting aspects of the tools from the perspective of the end user [16]. Our work differs from earlier ones in that it focuses on the implementation details of the tools, giving a greater emphasis to the needs of Grid middleware development. We set guidelines to decide the tools that can be included in this survey from a list of numerous products available. A previous survey shows that there were over 25 such tools at the time [2]. It is not possible to evaluate all tools in this paper due to time and space constraints. Some of the currently used tools are specialized on network monitoring, while others are analysis-only tools that can only work integrated with other components. We therefore confined our choice to a limited group of tools, selected by assessing their basic features and relevance to our evaluation criteria to be discussed shortly.

A general framework for a Grid performance monitoring system can be represented by the diagram shown in Fig. 2 [3]. This diagram shows the basic building blocks of an end-to-end monitoring system. One can use this framework as a reference for assessing how good a tool can be with regards to meeting this architectural requirement. An example of the above architecture is the Grid Monitoring Architecture (GMA) of the Global Grid Forum's Performance Working Group [6]. The GMA defines a set of requirements a Grid monitoring tool is expected to meet to be recognized as a recommended product for gathering, archiving and disseminating performance information. It also identifies three major components of this architecture: the *producer* of the monitoring information, the *consumer* and *directory* services. The *directory* service manages the producer and consumer components.

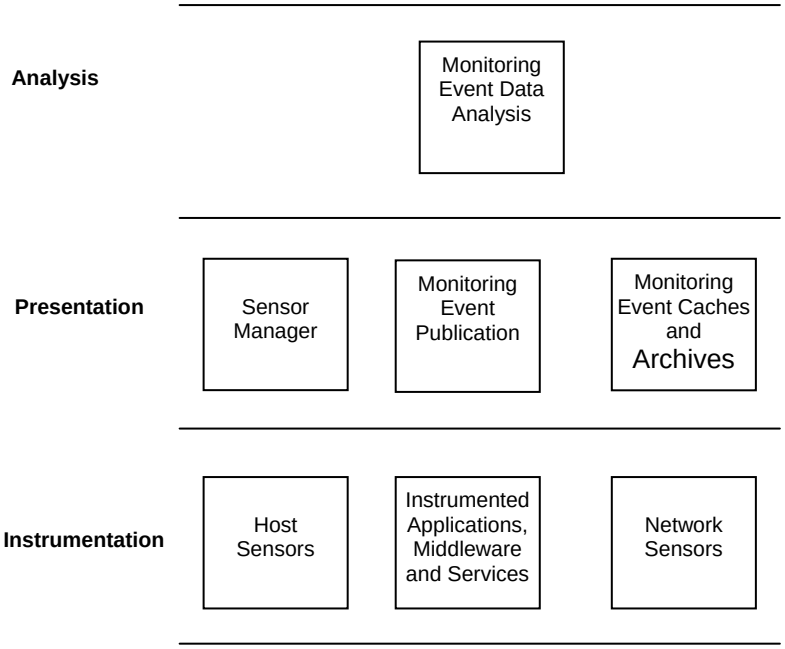

Fig. 1. Generic performance monitoring architecture.

Evaluation Criteria

This framework is prepared upon performance criteria that are likely required by our middleware. It considers the following factors:

1. Instrumentation Mechanism. Does the tool provide real time or offline instrumentation? To what extent does it allow perturbation of the application code?
2. Supported Languages by API. If one wishes to insert additional codes into an application for instrumentation purposes, which languages does the tool support? This factor takes into account the fact that the APIs should be compatible with the programming model used in simulation applications and MABS platforms.
3. Architecture. Does the tool meet the requirements defined in standard Grid monitoring architectures (GMA)?

4. Availability. Is the tool an open source or readily available?
5. Major Advantages. What are the strengths of the tool compared with similar tools, making a given too preferable over the others?
6. Specialization. Is the tool used to monitor all or limited aspects of the Grid environment? Some tools specialize in infrastructure monitoring, others in network and communication, while the remaining are focused on application monitoring. It is not usually common to find a tool that accommodates all the three.

5. Selected Tools

We started by examining the basic features of available tools based on the information provided by the developers of the tools. We then applied our guidelines discussed earlier and narrowed down the list to seven. Wherever possible, we studied their features from real time monitoring information available through their web sites. The selected tools are discussed below.

Ganglia is a hierarchically designed distributed monitoring tool for high performance computing systems. It has evolved into a global scale Grid infrastructure monitoring tool from a cluster level product. It employs two types of daemons, one for node level monitoring and the other for cluster level collection and aggregation of 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 [9]. Each node maintains monitoring data of all other nodes in a cluster.

GridICE implements the producer-consumer relations of the Grid Monitoring Architecture (GMA) through its five-layer structure comprising of a component doing the job of a producer at the bottom, and a consumer at the top [10]. The layers are mainly service-oriented, in line with the Open Grid Standard Architecture (OGSA) standards. GridICE is implemented with component software for each layer, having interactions with other performance monitoring tools and related Grid information services. The Measurement Layer is the bottom layer of this tool which contains the service responsible for gathering performance metrics data by probing into the Grid resources. The Publishing Layer makes the collected data available to

potential customers through a common interface and a directory based monitoring service. The other layers are used for archival of historical data, performance analysis and presentation of monitoring information.

Mercury is also a performance monitoring tool designed according to the specifications of the GMA [22]. It has a hierarchical architecture, with two types of monitors, local and main monitor. The local monitors are essentially producers whose sensors run on host machines and collect monitoring data. The main monitors collect sensor data and provide it to the consumer. The tool has a defined measurements metrics which the sensors use for host and application monitoring. A feature of Mercury is that it does not initiate take measurements unless requested by consumers. Mercury goes beyond GMA's requirements in that it supports application steering through its actuators which are used to control the behaviour of the monitored entity. This feature provides the user with a better control over running jobs such as application checkpointing, process migration and even restarting a crashed process. The control feature improves the execution environment by making Grid applications flexible and adaptive and giving users more control over their jobs.

NetLogger was initially developed for analysis of high performance distributed systems characterized by the transfer of large amounts of data at high speeds from a distributed storage server to a remote visualization client [12]. It provides features like bottleneck identification and network performance tuning by focusing on the elements of an application's end-to-end communication paths. With respect to application monitoring, NetLogger allows instrumentation code embedding into the application to be monitored. It logs time-stamped monitored event data into a buffer from where it is collected by its Monitoring Activation Service to be in turn sent to Monitoring Event Receiver which consumes the monitoring data, or Archive Feeder which converts events into SQL records for archival.

SCALEA-G is used for performance monitoring and analysis of Grid resources and applications [13]. It complies with the GMA and supports application and infrastructure monitoring. For application monitoring, it supports both source code and dynamic instrumentation of event monitoring and profiling.

The components of SCALEA-G are organized as services in conformity with the requirements of the GMA. The Sensor Services, which deal with measurement of application and system events, and the Client Service, which collects data from the Sensor, interact with the proposed middleware. The Client Service also handles performance analysis work and delivers the result to a presentation GUI, and mediates instrumentation of applications between the user and Source Code Instrumentation Service.

DIMEMAS was originally developed as a performance prediction simulator for message passing applications in parallel computing systems. Recently, it is extended to the Grid [15]. The DIMEMAS simulator can work on a uni-processor machine to predict the performance of an application running on a parallel system. To simulate the parallel execution, it utilizes data collected in a trace file from the parallel run (target parallel system under study) to reconstruct its time behavior. It can model the behavior of the target architecture with a set of parameters.

The trace of the real execution of an MPI application and the configuration of the target architecture are provided to the tool in two files. DIMEMAS then uses these files as its input to simulate the behaviour of the target and predict its performance. The trace file can be collected by running the application on any platform, not necessarily on the target system. The configuration file should however reflect the conditions of the target, such as a set of parameters that model the Grid, for instance. These parameters should show the size of the Grid and (number of machines running the application), the network characteristics such as connection type, local latency bandwidth, etc.

Prophesy is developed for performance analysis and modeling using application source codes in parallel systems originally. However, it has now evolved into a tool applicable in the Grid environment too. This tool uses different techniques for performance modeling of parallel applications written in C, Fortran 77 and Fortran 90. The instrumentation codes are inserted at various points in the source code, usually at the entry and exit of loops, functions and procedures. It has Data Collector, Database, Data Analysis, and GUI components [8].

The Data Collector deals with application profiling and instrumentation of the, the Database is used as a repository of performance and systems data, the Data Analysis component is concerned with model building and performance prediction, while the GUI handles the interaction with the user.

Prophesy assumes the application to be instrumented as a collection of modules, individual files containing the codes corresponding to a design entity. These modules are in turn composed of functions, which themselves are decomposed into basic units at the bottom of the hierarchy. It uses the raw application data to produce performance models. It employs automated modeling, adaptively improving the model over time. For performance analysis, it uses the method of kernel coupling, where the system attempts to generate the application model by studying the characteristics of its kernels (in this case function level). To generate analytical models of applications, it uses curve fitting and parametrization methods.

Table 1. Summary of Tools

Tool	Real-time Instrumentation	Supported Languages	Compliance to GMA	Availability	Major Advantages	Specialization
Ganglia	Requires modification of application source		Interface to GMA layers not clearly specified	Open Source, downloadable	Hierarchical, good data presentation and visualization.	Infrastructure Monitoring
GridICE	No support for application monitoring		Compliant can integrate with other tools.	Open Source, downloadable	Easier to integrate with Grid services such as schedulers	Infrastructure Monitoring
Mercury	Supports monitoring limited to predefined metrics	C++	Compliant	Open Source, downloadable	API allows both resource and application monitoring	Infrastructure, Application Monitoring
NetLogger	Supports both source and binary code perturbation	Python, C, Java	Implicitly GMA compliant	Open Source, downloadable	source code instrumentation API for several languages,	Communication Bottlenecks.
SCALEA-G	Supports both source code and binary code instrumentation.	Fortran, Java (byte code)	Compliant	On permission developer	Source code and binary level instrumentation ,	Application Monitoring
DIMEMAS	Offline simulation using trace data generated separately	Not Required	Not fully compliant	On Purchase, from developer	Offline tool, actual working environ-ment not needed, easy to implement	Application Monitoring
Prophesy	Real-time, supports source code instrumentation	C, C++, Fortran, HPF	Matches lower layers of GMA	Requires registration	Novel application modeling, perfor-mance prediction methods.	Application Monitoring

6. Conclusions and Future Work

A comparison of the monitoring support tools is presented in Table 1. One can find a longer list of analysis criteria in [2] in most aspects of performance monitoring. It can be realized that it is not easy to get all the desired features for our middleware from a single tool. We should therefore decide on the most appropriate tool for the intended application based on the specific needs.

Many of the tools examined have features such as hierarchical structure, analysis, visualization, measurement data repository for historical purposes, and provision of application instrumentation at the source and binary code level. One option would perhaps be to make use of the best combination of the tools fitting the operational scenario of the interested user.

GridICE, NetLogger and Mercury meet requirements such as use of common information model, support of web services standard interfaces and protocols. NetLogger and Mercury have API support for some programming languages,. The Mercury tool handles resource monitoring, an important input for performance modeling and prediction. The modeling and prediction layers of the middleware can therefore be supported by these tools.

An experiment is designed to study the proposed middleware architecture. A representative MABS application will be used as a test workload on a Globus Grid.

The application is basically a generic one, with simplified agent characteristics. The variables of interest in the experiment include: the number of agents taking part in a simulation, the complexity of computational work (granularity), and the interaction of the agents among themselves and with other services, the rate of communication between agents executing on different nodes. Application scalability with increase in the number of agents and the effect of computation-to-communication ratio will also be studied to understand the performance characteristics of MABS applications in the Grid environment.

To facilitate performance monitoring, we add API extensions and instrumentation codes at critical points in the source code of the prospective application. The performance of MABS applications with and without the support of the middleware should then be compared to evaluate the usefulness of the proposed middleware architecture.

Based on performance data received from the monitoring tools, the middleware should be able to carry out application modeling and performance

prediction, and optimize the job submitted to schedulers to match the available resources.

There are different strategies that one can apply to optimize executable jobs and improve performance in similar situations. Some of them are:

- Controlling the granularity of the codes in accordance with an acceptable computation-to-communication ratio;
- Managing the launching of agents in such a way that those with intense communication are located on the same node so that their communication does not introduce network traffic;
- Reorganizing the components of the MABS platform in such a way that the interaction between simulation agents and the platform services does not degrade performance severely.

The middleware should apply one or more of these strategies dynamically and adaptively to tune the application. The extent to which performance improvement can be achieved should then be investigated experimentally.

We believe that the outcome of such an experiment assists the ongoing work to use the Grid for MABS applications. Developers of other domain-specific applications and Grid middleware will also benefit from this work.

7. References

[1] J.C. Browne. *"A Critical View of Computer Performance Evaluation."* Proc. 2nd International Symposium on Software Engineering, 1976.

[2] M. Gerndt et. al. *"Performance Tools for the Grid: State of the Art and the Future."* White Paper. Automatic Performance Analysis: Real Tools (APART). www.lpds.sztaki.hu/zsnemeth/apart/repository Visited March 2005

[3] I. Foster, C. Kesselman. *"The Grid2 Blueprint for a New Computing Infrastructure"*. p 319-51. Morgan Kaufman Publishers, San Francisco, 2004.

[4] R.M. Bada et. al. *"Performance Prediction in the Grid Environment. "* Proc. First European Across Grids Conference, Spain 2003.

[5] M. Baker and G. Smith. *"GridRM: A Resource Monitoring Architecture for the Grid."* Proc. Third International Workshop on Grid Computing, Baltimore, MD, USA, November 2002

[6] B. Tierney et. al. "*A Grid Monitoring Architecture.*" Grid Forum Working Draft GWD-GP-006-1, February 2001. http://www.gridforum.org. Visited March 2003.

[7] J. M. Schopf, J. Nabrzyski, J. Weglarz. "*Grid Resource Management.* " (pp 15-41), Kluwer Academic Publishers, Boston 2003.

[8] X. Wu, V. Taylor, R. Stevens. "*Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications.*" ACM SIGMETRICS Performance Evaluation Review. Volume 30, March 2003. ACM Press USA. ISSN:0163-5999 p 13-18

[9] M. Massie, B. Chun, D. Culler. "*The Ganglia Distributed System. Design, Implementation and Experience*." Parallel Computing, v 30, July 2004, p 817-40.

[10] S. Andreozzi et. al. "*GridICE: A Monitoring Service for Grid Systems.*" Future Generation Computer Systems, v21, n4, April, 2005, High-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project, p 559-571

[11] X. Zhang, J.L. Freschl, J.M. Schopf. "*A performance study of monitoring and information services for distributed systems.*" Proc. 12th IEEE International Symposium on High Performance Distributed Computing, 2003, p270-81 ISBN: 0769519652.

[12] B. Tierney et. al. "*The NetLogger Methodology for High Performance Distributed Systems Performance Analysis.*" Proc. IEEE High Performance Distributed Computing conference, July 1998, LBNL-42611

[13] H. Truong and T. Fahringer. "*SCALEA-G: A Unified Monitoring and Performance Analysis System for the Grid.*" Second European AcrossGrids Conference, AxGrids 2004. Revised Papers (Lecture Notes in Computer Sc. Vol.3165), 2004, p 202-11

[14] D. Gunter et. al. "*NetLogger: A Toolkit for Distributed Systems Performance Analysis*." Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728), 2000, p 267-73

[15] R. Badia et. al. "*DIMEMAS: Predicting MPI Applications Behaviour in the Grid Environment.*" First European Across Grids Conference. Revised Papers (Lecture Notes in Computer. Science. Vol.2970), 2004, p 257-64.

[16] M. Broberg: "*Performance Prediction and Improvement Techniques for Parallel Programs in Multi Processors.*" PhD Thesis, Blekinge Institute of Technology, 2002.

[17] Z. Zhang et. al. "*Multi-agent System Solutions for Distributed Computing, Communications and Data Integration Needs in the Power Industry.*" IEEE Power Engineering Society General Meeting, 2004. Vol.1 p 45 - 49

[18] G. Weiss. "*Multi Agent Systems A Modern Approach to Artificial Intelligence.*" The MIT Press, London. 1999.

[19] S. A. Jarvis et.al. "*Performance Responsive Middleware for Grid Computing*." *AHM2003: Delivering e-Science, Nottingham,*, September, 2003.

[20] "*GridLab. A Grid Application Toolkit and Testbed*." Grid Monitoring Specifications http://www.gridlab.org/Work_Packages/wp-11/index.html Visited August 2005

[21] Bradshaw, J.M. "*An Introduction to Software agents*" MIT Press, Cambridge USA. 1997. pp. 346.

THREE

Applications of Agent Based Simulation

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

School of Engineering, Blekinge Institute of Technology
Ronneby, Sweden

Abstract

This paper provides a survey and analysis of applications of Agent Based Simulation (ABS). A framework for describing and assessing the applications is presented and systematically applied. A general conclusion from the study is that even if ABS 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 ABS often is not utilized. We illustrate this by providing some concrete examples. Another conclusion is that important information of the applications, in particular concerning the implementation of the simulator, was missing in many papers. As an attempt to encourage improvements we provide some guidelines for writing ABS application papers.

1. Introduction

The research area of Agent Based Simulation (ABS) continues to produce techniques, tools, and methods. In addition, a large number of *applications* of ABS have been developed. By ABS 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 ABS applications (as they are described in the papers) and to identify trends, similarities and differences, as well as issues that may need further investigation.

As several hundreds of ABS applications have been reported in different publications, we had to make a sample of these. After having performed a

preliminary search for papers describing ABS 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%. We then chose to focus on the MABS publication series and found 28 papers containing ABS 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 ABS 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

An ABS application models and simulates some real system that consists of a set of entities. The ABS 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 agents. We will use the terms “system” and “entity” when referring to reality and “multi-agent system” and “agent” when referring to simulation models. For each paper we describe different aspects of 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, the intended end-user, and the purpose of the ABS application.

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* consists 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* consists 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 other 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.

End-users: The end-users of an ABS application are the intended users of the simulator. We distinguish here between four types of end-users: *scientists*, who use the ABS in the research process to gain new knowledge, *policy makers*, who use ABS for making strategic decisions, *managers* (of a systems), who use ABS to make operational decisions, and *other professionals*, such as architects, who use ABS in their daily work.

Purpose: The purpose of the studied ABS 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.

2.2 Modeling Approach

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

Simulated Entities: They are the 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.

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.

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.

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.

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.

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. 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. Whereas, in a recursive structure, entities are arranged in groups, where the organization of each group could be in either of the forms above, and these groups are interacting among each other to accomplish their tasks. The three types of MAS structure are illustrated in Fig 1.

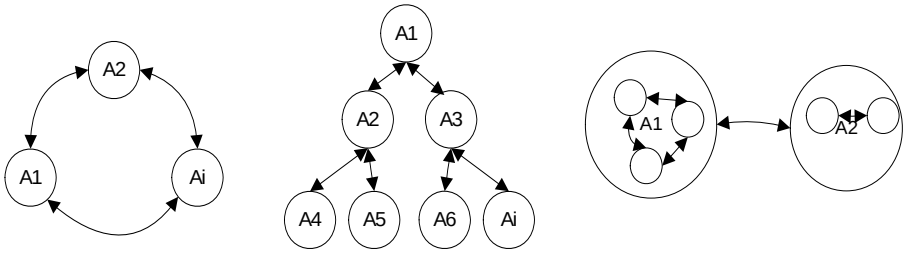

Fig. 1. *Peer-to-peer, hierarchical, and recursive organization of a MAS.*

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 the following aspects:

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

Simulation size describes the number of agents participating in the implementation of the ABS application. If the number is different between

simulations or is changing dynamically during a simulation, we will use the largest number.

Scale: The size of data used in the actual simulations 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.

Input data: 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.

Distributed: ABS 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 several computers in a distributed environment, if the number of agents is large.

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.

Maturity: ABS applications can have varying degree of maturity. In our 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* where the application has been tested in a laboratory environment. The final level, *deployed system*, indicates that the ABS system actually is or has been used by the intended end-users, e.g., traffic managers that use a simulator for deciding how to redirect the traffic when an accident has occurred. If the authors of the paper belong to the intended end-users (researchers), we classify the application as deployed if the authors draw actual conclusions from the simulation results regarding the system that is simulated (rather than just stating that ABS seems appropriate).

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 ABS 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.

Validation: In order to confirm that an ABS 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.

3. Results

In table 1 the framework is summarized. Table 2 shows how the papers were classified according to the framework. If a paper 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 "-".

Table 1. *Summary of the framework*

	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. Transport/traffic systems
	End-user	1. Scientists 2. Policy makers 3. Managers 4. Other professionals
	Purpose	1. Prediction 2. Verification 3. Analysis 4. Training
Modeling approach	Simulated entity	1. Living 2. Physical artefact 3. Software process 4. Organisation
	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.
	Simulation size	1 - 10.000.000
	Scale	1. Limited/partial 2. Full-scale
	Input data	1. Artificial data 2. Real data
	Distributed	1. no 2. yes
	Mobile agents	1. no 2. yes
Results	Maturity	1. Conceptual proposal 2. Laboratory experiment 3. Deployed
	Evaluation	1. None 2. Qualitative 3. Quantitative
	Validation	1. None 2. Qualitative 3. Quantitative

Table 2. The classification of the studied papers.

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

4. Analysis

4.1. Problem description

The results indicate that ABS is often used to study systems involving interacting human decision makers, e.g., in social, organizational, economic, traffic and transport systems (see Fig. 2). 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.)

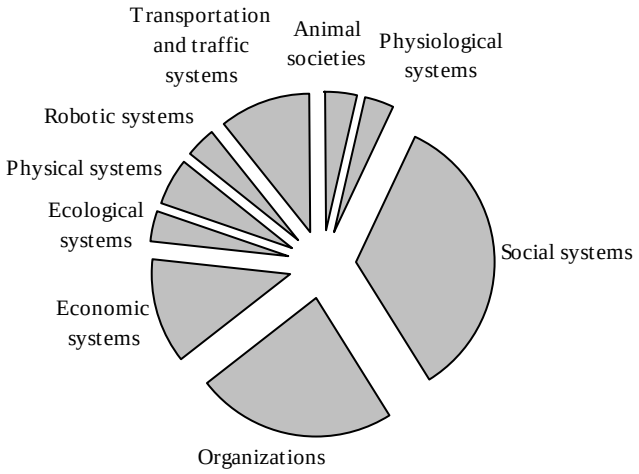

Fig. 2. *The distribution of the type of domains simulated.*

In more than half of the applications, researchers were the intended end-user. As can be seen in Fig 3., the most common purpose of the applications included in the study was analysis. However, no paper reported the use of ABS for training purposes indicating that this may be an underdeveloped area.

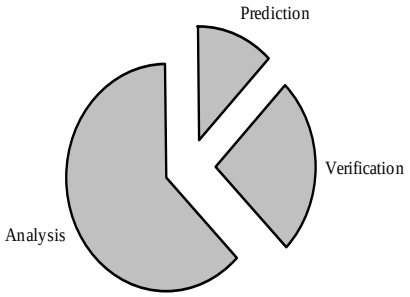

Fig. 3. *The distribution of purpose.*

4.2. Modeling Approach

The simulated entities are mostly living things, indicating that ABS 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%).

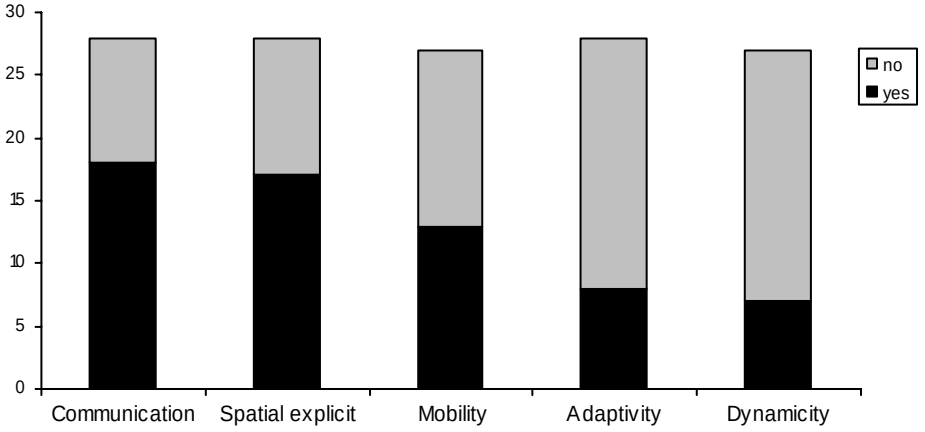

Fig. 4. The distribution of modeling aspects.

However, as illustrated in Fig. 4, 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 ABS. 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 ABS tools and platforms make limiting assumptions regarding the way that entities are modeled. The number of agents in the simulation experiments is typically quite small (see Fig. 5). 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. 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. 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.

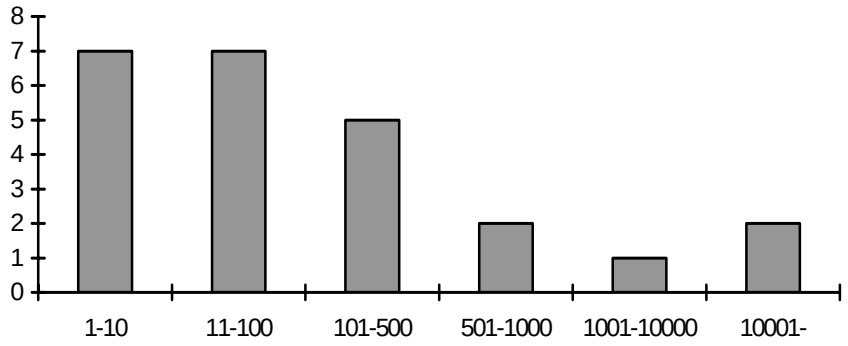

Fig. 5. *The frequency of different simulation sizes (number of agents).*

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. 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 ABS applications that are reported to be deployed to solve actual real world operational tasks. The examples of deployed systems are limited to the cases where the researchers themselves are the end-users. The cause of this could be the fact that ABS is a quite new methodology, or that the deployment phase often is not described in scientific publications. As illustrated in Fig. 6, 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. Also, comparisons to other approaches are very rare.

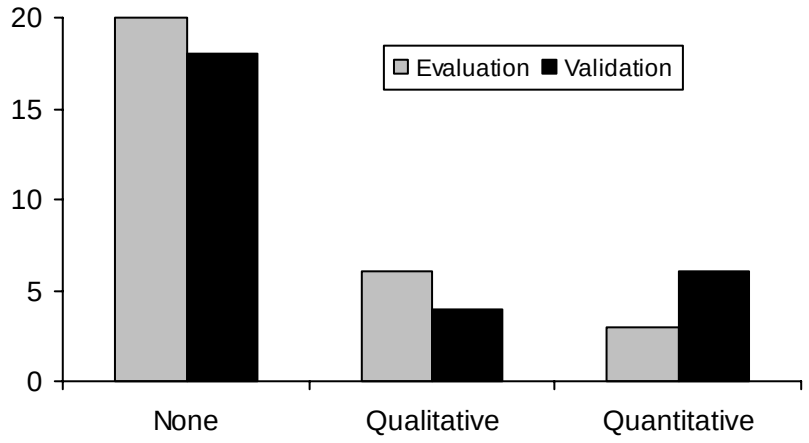

Fig. 6. *The frequency of different types and evaluation.*

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 ABS area. 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 ABS 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 ABS tools and platforms are seldom used and instead the simulation software is developed from scratch using an 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 ABS. 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 ABS 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
    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 by 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, in order to reduce any bias. Another interesting study would be to make 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. J.S. Sichman, L. Antunes (Eds.), *Multi-Agent Based Simulation VI*, LNAI Vol. 3891, 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 Marke*t", 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, "T*he 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. V. Furtado, A. Melo, M. Belchior, "*Analyzing Police Patrol Routes by Simulating 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]

FOUR

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

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

Abstract

The Grid 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 for these applications 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 in the Grid environment.

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

1 Introduction

The Grid has emerged as an important infrastructure to support complex applications in various fields. Applications 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, they are limited, partial, conducted on a smaller scale, with fewer agents, making use of many simplifying assumptions, etc [2]. Most MABS models 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 with 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 controlled experiments to study the behaviour of MABS, investigate application design issues and ways of overcoming performance bottlenecks. We also developed a prototype of the middleware 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, followed by a discussion on the achieved performance gain using a prototype of the middleware, in Section 7. 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 massive 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 is 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. It is convenient to implement large-scale simulations as distributed parallel applications; a parallel distributed computing system such as the Grid can thus be a natural working environment.

Performance insensitive MABS applications are not good candidates as a workload for the Grid. In fact, a MABS designed without due regard to performance issues does not utilize computing resources efficiently and can thus be a liability to a shared execution environment. 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 efficiency. A MABS platform supported by a middleware that can 'sense' the execution environment and adapt the application for best performance on the Grid is desired. To achieve this, a performance-aware middleware that works with the MABS platform to support dynamic application tuning and optimization for the Grid is required.

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 for distributed network 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 complex scheduling problems.
2. *High communication-to-computation ratio*. As explained earlier, agents simulating real world entities should mimic the interactions between the entities they model. The interaction is inter-agent messaging, where the origin and destination agents in a message may be 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 should not perform any action (hence 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, the forced waiting can undermine performance.

3. *Time-step synchronization*. For the simulation to reflect the chronological sequence of events in the simulated problem and guarantee causality of actions, the time frame of the agent executions throughout the simulation must be managed centrally. This normally requires forcing a currently running thread to yield, 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 status. 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. This is due to the fact that the course of actions (thus sequence and complexity of executable program codes) pursued by agents in a MABS is not predetermined, rather depends on the instantaneous decisions taken by individual agents. 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 contain important 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.

The implementation of performance analysis techniques in the context of Grid computing is more complex than in other environments, due to the dynamic behaviour of the Grid and the heterogeneity of resources. Because of this, existing 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 therefore arises from this inherent nature of the Grid environment

4. Proposed Architecture

The proposed Grid middleware is to be used as an interface between a multi agent based simulation (MABS) platform and the Grid resources and services. As explained earlier, the main 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.
- Reallocating 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 interfaces 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. The Grid scheduler will then be able to receive an optimized job, matching predicted performance and execution requirements. The figure below shows the architecture of the proposed performance aware middleware.

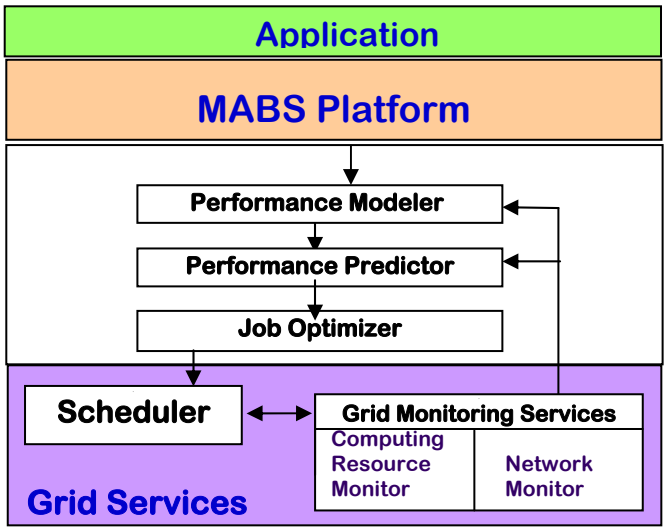

Fig. 1 *Proposed Architecture of Middleware. The Grid monitoring services collect data from components of the Grid.*

The middleware consists of the following components:

1. *Performance Modeler*: this component uses performance data collected through Grid monitoring services to build the execution performance model of the simulation workload. The model contains information needed to make scheduling decision, reallocation strategy, etc.
2. Performance Predictor: this component predicts the execution time of the simulation based on the output of the performance modeler and data from Grid monitoring services on the current status of the Grid infrastructure. The prediction is bound to change with changes in the Grid environment.

3. Job Optimizer/ Match Maker: undertakes the necessary modifications on run-time parameters of the executable 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. The figure below shows the components in detail.

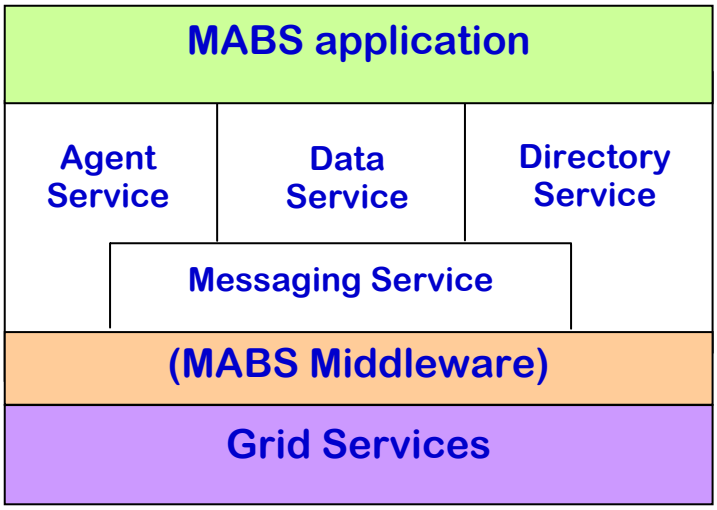

Fig. 2 *Major components of a MABS application platform in the Grid environment*

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

- Launching and termination of agents;
- Execution of agent code;
- Registration and deregistration of agents to platform;
- Enforcement of time-step synchronization, 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. At the time of this study, it was not possible to find any MABS platform deployed in the Grid environment. We could not find mature products other than those at preliminary stages [4] and conceptual frameworks such as the one presented in [3], which are yet to be implemented. We had to therefore design an experimental MABS platform and application from scratch based on the framework of figure 2, as a synthetic simulation workload incorporates the prominent MABS 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 supported by the Messaging Service.
- Has a computational component that implements the behaviour of the real world entities it represents.

For the purpose of this experiment, the agent code will have two components, computational and communication, 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 messaging activities. The actions of an agent are therefore realized by a thread code that performs a computation, followed by a messaging and yields control of the CPU to enforce 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 will essentially be 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 these (communication and synchronization) on performance for different levels of agent 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 agent 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 by taking measurements with different inputs and repeating the experiment several times. It is seen 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 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 were connected via a 100Mbps switch to the Internet. Since the main purpose of the study was 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 feature explained earlier. The Grid nodes did not thus have additional loads. Furthermore, some components of the Globus toolkit such as GFTP and security features, although installed, were not used here since their presence or absence bears no relevance to this experiment.

Each Grid node hosts an identical copy of the MABS platform deployed as a set of Web Services in its Globus container. Another machine used as a master node (also called the Grid client) serves as a launching pad to execute the simulation. On this machine, 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 in both environments are functionally the same, we discuss the Grid version in the rest of the paper.

5.3. Implementation of Workload

The workload is a Java multi-threaded application written in accordance with the requirements of the 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 container where the MABS platform is hosted. 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 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. Because all agents take control of the CPU turn by turn, the expected reply should be received anyway.

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 stays 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. It is given by the number of messages sent out by an agent to agents residing on nodes other that hosting the sender, out of every 100 messages sent by the same 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 synchronization) in this way, the master node will send them a command to proceed with the next time step or computational loop through a web service invocation. This synchronization process is repeated for a specified number of times given by the ratio of the size of the workload to that of the computation performed in one loop.

The client machine launches the application by initializing the simulation parameters and invoking Web services on each Grid node, passing the required parameters of the simulation to the nodes. This machine also monitors the overall execution of the application and facilitates the delivery of outbound messages in cross node communications through the Agent Service.

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.

6.1. Effect of Multi-threading on Performance

If the program code of individual agents is 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 non-linearly. To illustrate this, repeated 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 with no outbound communication (OB=0%) and task granularity of 5ms is shown in the following figure.

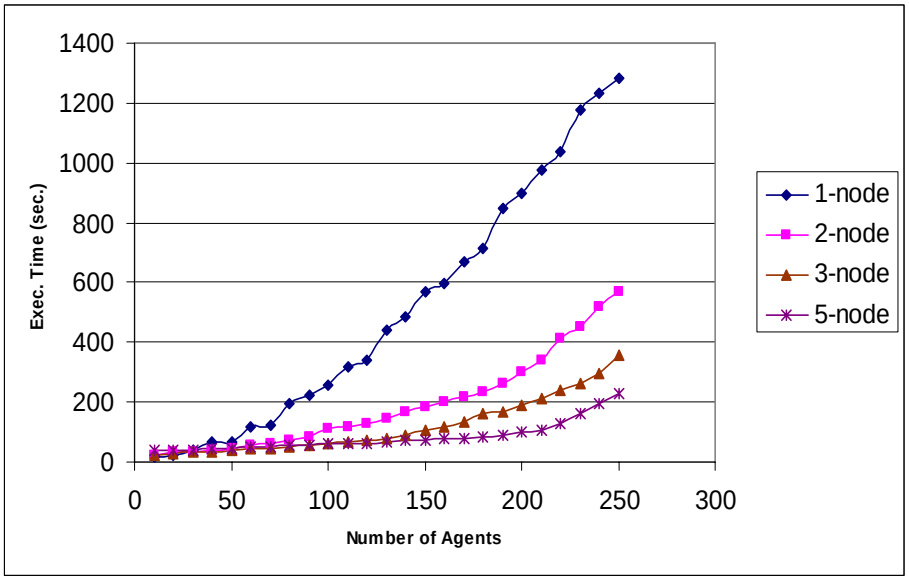

Fig.3. Execution time of MABS applications on the Grid increases non-linearly with simulation size.

As can be seen from the plot, implementing the agents as individual threads causes the execution time to increase in a quadratic fashion with the increase in the number of agents. This factor is a source of significant degradation in performance. It also 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 M^2. 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 one of the requirements of the simulation is that agents expecting a reply for the message 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 expected *reply* arrives. The contribution of this waiting time to the overall execution time could even be much higher 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 when the size of each inter-agent message is too large to fit into the communication buffers. The network latency may then be the dominant contributor to performance degradation.

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. However, time step synchronization may affect performance by introducing additional overhead as can be seen in Fig. 4.

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 (behaviour of local operating system, implementation of Java virtual machine, etc) take a larger share of the performance drop.

In the plots shown in Fig. 4a, the task granularity is 2ms (corresponding to fine), while it is 200ms (corresponding to coarse) in Fig. 4b.

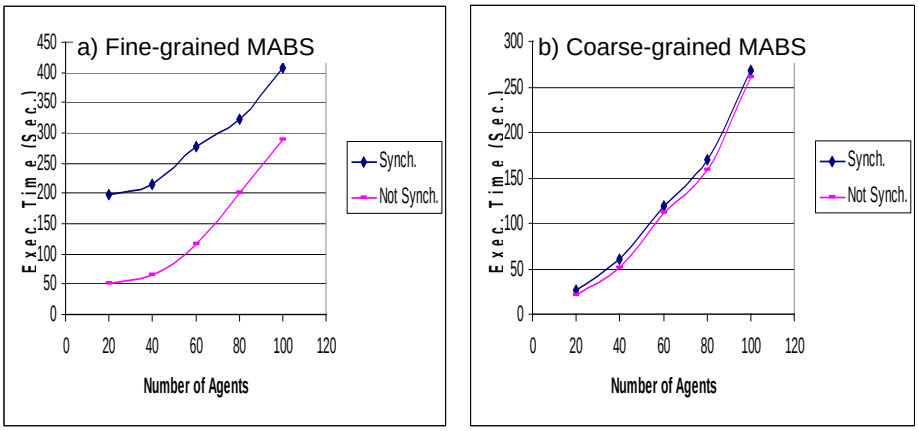

Fig. 4.Comparison of simulations with and without time-step Synchronization on a 5-node Grid, with no outbound communications. Fine grained simulations are affected most.

In limited cases, MABS applications may not need any time-step synchronization at all. Such cases arise when the simulated system does not involve any communication between entities, or the chronological sequence of the simulated events is of no interest. In such cases, it is possible to achieve significant performance gains time-step synchronization is not a functional requirement of the simulation.

7. Middleware Support for Performance Improvement

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 in this Section.

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 peers 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 deployed 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. The algorithm used in agent reallocation is summarized as follows:

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.

In this experiment, it is assumed that the list of peers an agent communicates with does not change with time (static random). 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 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:

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

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

In the experiment run numbers 1, 2 and 3, prior to reallocation, each agent had 32 peers of which only 20% are executed on the same node. Therefore, 80% of the messages were outbound. After reallocation, it was possible to have as many as 28 peers on the same node. The outbound communication will thus be much lower. In runs 4 and 5 however, after reallocation, it was managed to deploy only about half of the peers on the same node. Due to the reallocation, we expect some performance gain in the form of reduction in network latency. The magnitude of this gain depends mainly on the size of individual messages relative to the communication buffer size at the nodes.

8. Conclusions and Future Work

The experiments have investigated some of the 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.

Grid based platform using a performance-aware middleware can support fine tuning of the overlying application if the simulation size, the communication characteristics and the granularity of the application can be determined dynamically. Simulation modelers can also decide on the architecture of the simulation, complexity of the agent code, etc., anticipating the performance benefits that can be achieved by designing an efficient simulation.

Since a production Grid consists of heterogeneous resources in a dynamic environment, the use of the middleware in more realistic scenarios needs to be further studied. A practical simulation application having the prominent features of MABS should be used as a workload. The prototype middleware should be expanded to incorporate additional features, such as determining application granularity and the degree of its influence on performance. If the MABS application characteristics can be determined by the middleware, it will be possible to build a performance prediction model and facilitate scheduling.

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.

9. 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., Persson, M.: "*Applications of multi-agent based simulations.*" Seventh Workshop on Multi-Agent-Based Simulation, to be published by Springer in the Lecture Notes in Computer Science series (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.: "*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)

5. Baker, M., Apon, A. et al.: "*Emerging Grid standards.*" IEEE Computer Press, USA, Vol. 38 No. 4 (2005): 43-50.

6. Sotomajor, B.: "*The Globus toolkit 4 programmer's tutorial.*" http://gdp.globus.org/gt4-tutorial/. Visited March 2006.

7. Helsinger, A. et al. "*Tools and techniques for performance measurement of large distributed multi-agent systems.*" AAMAS'03 (2003) Australia

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. et. al.: "*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-verlag Berlin Heidelberg pp. 207-214, (2004)

15. Marieto, M.B. et. al.: "*Requirements Analysis of Agent-Based Simulation Platforms.*" In: Jaime Simão Sichman, François Bousquet, Paul Davidsson (Eds.): Multi-Agent-Based Simulation. Lecture Notes in Computer Science, Vol. 2581. Springer-Verlag Berlin Heidelberg (2003).

FIVE

Performance Prediction of Multi-Agent Based Simulation Applications on the Grid

Dawit Mengistu, Lars Lundberg, and Paul Davidsson
Department of Systems & Software Engineering,
Blekinge Institute of Technology, Ronneby, Sweden
{dawit.mengistu, paul.davidsson, lars.lundberg} @bth.se

Abstract

A major requirement for Grid application developers is ensuring performance and scalability of their applications. Predicting the performance of an application demands understanding its specific features. This paper discusses performance modeling and prediction of multi-agent based simulation (MABS) applications on the Grid. An experiment conducted using a synthetic MABS workload explains the key features to be included in the performance model. The results obtained from the experiment show that the prediction model developed for the synthetic workload can be used as a guideline to understand and to estimate the performance characteristics of real world simulation applications.

Keywords - Grid computing, Performance modeling, Performance prediction, Multi-agent simulation.

1. Introduction

The computational Grid has emerged in the past years as a key technology, an execution environment for applications with huge resource requirements. One of the fields where the Grid can provide a unified and powerful execution

environment is simulation. Large scale Multi-Agent Based Simulation (MABS) applications facilitate the study of complex scientific problems if they can be deployed in a distributed computing environment with sufficient resources.

A major challenge to implementing MABS on the Grid is application performance. MABS applications have unique features distinguishing them from other applications executed on the Grid. These features are not as such trivial and their impact on performance is tremendous. There is thus a need to study and understand their individual or combined effects.

One of the features commonly recognized as a cause of performance degradation in MABS applications is the high communication-to-computation ratio resulting from interaction between agents participating in the simulation. Maintaining coherence and causality of events in the simulation by matching the flow of application execution with the sequence of events in the real world (time-step synchronization) is also another problem. Another aspect of the time problem is that the granularity of the application code executed (the task executed by the agent in a unit of time) with in one unit of simulation time is not always fixed. As is well known, it would be difficult to develop an accurate performance prediction model for such applications. The purpose of this study is therefore, to gain better understanding of the performance issues and to develop a prediction model for load balancing and scheduling decisions.

In this paper, we present a study of MABS application characteristics and a performance prediction model. Using a synthetic MABS application as a workload, an experiment is conducted to investigate the relationship between the MABS application architectural features and the run-time behaviour. To verify the validity of the developed model, the experiment was conducted with a workload that has the features of a real MABS application. It is observed that our models provide a better understanding of the underlying process and assist in the projection/prediction of the performance of similar MABS applications. The rest of this paper is organized as follows: a brief overview of MABS applications and the motivations to use the Grid for MABS is presented next. The methodology applied in this work is then discussed, followed by the experimental setup. We then analyze the measurement data from the experiment and discuss the results. We conclude the paper by summarizing the findings of our experiment and citing directions for future work.

2. MABS ON THE GRID

2.1 Multi-Agent Based Simulation

MABS is a methodology used to study and understand the dynamics of real world phenomena in domains which involve cooperative problem solving. The players in these domains are characterized as entities having autonomous and social behaviour. The real world entities are modeled as software agents that interact with each other to achieve a common objective. In large scale problems, the number of these entities is very large (millions, in some cases).

A large scale simulation requires the availability of adequate computing power and a distributed MABS platform capable of utilizing the availed resources. The Grid offers a robust distributed computing infrastructure needed for simulations of this type. In order to deploy MABS applications on the Grid, one would need to understand the important features of these applications affecting performance.

2.2 MABS Features

A detailed list of the features characterizing MABS applications is discussed in [2]. As some of these features bear no impact on the performance of the application per se, we have omitted them from this discussion.

1. *Multi-threading*. The autonomous behaviour of each simulation agent can be modeled using a program code executed within its own thread of control. Large-scale simulations involve a large number of agents and hence, a high level of multi-threading. The effect of this on performance is manifested in terms of excess thread setup time at the launching of the simulation and massive context switching overhead throughout the course of the simulation. It may also cause scheduling problems, the extent of which depends on the run-time environment and operating system behaviour.
2. *High communication-to-computation ratio*. The 'social' behaviour of agents embodied in MABS requires that agents communicate with each other or with information directory services to update their knowledge (beliefs) and accomplish a collective task. If the MABS is deployed as a distributed application on several nodes, the communication may involve

frequent cross node data exchange and lengthy idle time for the communicating agents if the next course of action they take depends on the content of an anticipated message.

3. *Time*-step *synchronization*. For the simulation to reflect the chronological sequence of events in the real world problem and guarantee causality, a central reference clock needs to be established. The interval between successive 'ticks' of the clock is the smallest possible indivisible period of time for the simulation [18]. The execution of the simulation advances in steps of this unit of time. Each agent executes some task (defined as granularity) corresponding to what the simulated entity accomplishes in the real world between two successive clock ticks. Since two real world tasks accomplishable with in the same unit of time do not translate into application codes of equal size, the granularity of the application will be variable. Agents which have a short task for a given period finish their job before their assigned quantum expires (voluntary yield of thread) while those with a long task may not be able to finish their part with in one quantum of CPU time. The yielding thread will stay in the wait state until all other agents execute their tasks and be at a similar temporal status.

These factors usually occur in combination making the MABS application complex. Deploying such applications on a dynamic heterogeneous environment like the Grid will make the problem even more complex. Performance modeling of a MABS application for prediction, load balancing and scheduling purposes should therefore make due consideration of these features.

2.3 The Grid

The Grid can offer the proper execution environment for MABS applications for the following reasons:

1. As explained earlier, the computational resources needed for certain simulations are so large that they cannot be effectively run on small systems. Simulations may require that thousands or even millions of agents perform highly complex and data intensive tasks.

2. Since MABS grew out of distributed artificial intelligence, a distributed computing platform like the Grid serves as a natural environment to plan, develop and implement agent based simulations.

3. Methodology

3.1 Workload Modeling

The workload is a multi-threaded application, each agent has its own thread of execution and communication capabilities, and a program code that captures the roles and objectives of the real world entity modeled by the agent. In common multi-agent applications, the agents execute the task defined in their roles and perform a messaging operation. The agents update their beliefs about the environment, and exchange information with their peers during this messaging time. The agent code has thus separate computational and communication components executed in a cycle. Coincidentally, this arrangement is useful for our work, to study the earlier discussed features of MABS. An agent thread thus performs a computation, followed by a messaging and yields control to enforce the time-step synchronization.

If the application is deployed on a stand-alone machine, since all threads run on that same machine, the inter-thread communication is essentially data movement with in the same physical memory. On the Grid, however, the threads are launched on separate machines and the communication involves transfer of data over the network too.

The realization of the Grid workload furthermore involves partitioning the simulation into equivalent tasks, with each task to be launched on a node. The 'social' relations in the physical entities involve data transfers within the same machine (*inbound* messaging) or across nodes (*outbound* messaging). The terms inbound and outbound refer to the destination of the messages with respect to the location of the sender. 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 study the effect of time-step synchronization by defining the application's granularity as another parameter.

The workload is implemented both as a single-node (standalone) version and a Grid application to evaluate the performance.

3.2 Performance Metrics and Measurement

We have identified the following quantities of interest to be included in our performance metrics:

- Simulation run time
- Thread setup time
- CPU utilization

Data manipulation and transfer tasks were deferred to the end of the workload execution not to interfere with the normal course of the application and to have as little perturbation as possible to the instrumented code.

In the initial experiment, the workload model consists 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 [13].

The validity of the experimental setup was verified to ensure that the workload and performance metrics chosen indeed conform to the envisaged objective and are equally valid for both the stand-alone and Grid environments. One of the factors important to judge the validity of the experiment is its repeatability.

3.3 Performance Modeling

In general, a performance model should take into account the effect of the various sub systems such as processor, memory, disk, network, software efficiency, algorithms, etc. In this experiment, however, the study shows only the application’s behaviour, i.e., the effect of communication and task granularity.

The envisaged performance model will thus have the following parameters as inputs:

- the number of agents (N),
- the number of Grid nodes (M)
- the granularity (G), and
- the outbound communication rates (OB).

Measurements are taken from the system for several runs with different values of these parameters. Since the outbound communication is not applicable in standalone environment, it will appear in the Grid version only. Furthermore, the number of nodes in a stand-alone configuration is equal to one (M=1).

The performance model computes the execution times T_{st} and T_{Grid} of simulation runs in stand-alone and Grid environments respectively, as a function of the above parameters:

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

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

T_{setup} is the time required to fully launch the simulation. This time depends on the number of threads 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 signifying 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 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):

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

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:

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

f_i is a function that 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 are then analyzed using the MATLAB software to build the required performance models. The obtained models will be validated with more measurements to be compared with the predictions.

3.4 Implementation

The stand-alone application version is a java multi-threaded application written in accordance with the requirements of the workload model explained in III.a above. On the Grid, it is implemented as a web service application launched from a master (client) node where the workload is partitioned into pieces of balanced sizes distributed to worker nodes.

The Grid version of the experiment was conducted on a Globus (GT4) Grid testbed with Linux machines having PIII 1000MHZ, 512MB RAM, connected via a 100Mbps switch to the Internet. The stand-alone version was run on a Linux machine with identical configuration, OS environment and load conditions.

As explained earlier, the simulation space is evenly divided into the number of Grid nodes on which the MABS workload is to be run. 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, each node will host 50 of the agents. Communications can take place between agents residing on the same or different nodes, depending on the requirements of the real world problem.

The agents are instantiated as individual threads in the Grid web service when the client node launches the application. 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 arrive any way.

Since the threads need to maintain coherence with respect to the simulated time, they should 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.

The ratio of outbound messages to the total number of dispatched messages is an important parameter to study the effect of network latency. It is defined as *outbound communication ratio*, OB, the ratio of the number of messages sent out by an agent to agents residing on other nodes, out of every 100 messages initiated by that agent.

To launch the application, the master code on the client machine 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, collects performance data, and facilitates the delivery of outbound messages in cross node communications.

Several runs of the experiment were conducted with different values of the model parameters explained earlier, to understand their individual and combined effects. Analysis of the measurement data and the performance predictor obtained from it, are discussed in the next section.

4. Results

4.1 Performance Modeling with Synthetic Workload

The execution-time versus number-of-agents relationship is non linear. Analysis with MATLAB shows that the curves fit to quadratic equations more accurately. The performance model will therefore use the value $m=2$ in equation (2) presented in Section III. One possibility to use a linear model is to divide the plot into several piecewise linear sub regions to have an accurate linear model for a region of interest. The problem with this approach is however, that the model will be applicable to only a narrow range of N and models should be obtained for different sub regions to make it useful.

Analysis of measurement data shows that it is difficult to find a single equation model in (N,M, G, OB) that can be used universally. We therefore obtained separate models for different granularity values as follows.

$$T_g(N) = f_2(M,OB)N^2 + f_1(M,OB)N + f_0(M,OB) \quad (4)$$

The subscript *g* refers to the granularity in milli seconds. For illustration, the performance model for granularity of 10ms is shown in equation 5.

$$\begin{aligned} T(N) = {} & ((-0.0024*M) + (0.0001*OB) + 0.0113) * N^2) + \\ & ((-0.2748*M) + (0.020*OB) + 1.1535)*N) + \\ & ((75.2585*M) + (2.7441 * OB) - 113.757) \end{aligned} \quad (5)$$

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

$$T_{setup} = (0.000476 * N^2) - (0.03752 * N) + 8.2625 \quad (6)$$

The coefficients of OB are small indicating that the effect of outbound communication is minimal for large N (larger simulations). In large simulations, since the threads spend more time waiting in a queue than in actual execution, the network latency component may not be the dominant component of the over all execution time. The coefficients of M are negative for the linear and quadratic terms, which means running the simulation reduces execution time for large N. The models are validated with additional measurements as can be seen in the data given in tables 1 and 2. The chosen values of the inputs are selected from the model's valid ranges.

G (ms)	M	OB (%)	Response Time (s)		
			Predict.	Measured	Dev. (%)
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 model: n=180

G (ms)	M	OB (%)	Response Time (s)		
			Predict.	Measured	Dev. (%)
2	2	10	3516	3025	13.9%
2	4	10	1874	1572	16.1%
2	5	10	1053	1210	14.9%
5	2	20	2195	1914	12.8%
5	4	20	984	971	1.3%
5	5	20	953	879	7.8%
10	2	30	2087	1761	15.6%
10	4	30	756	790	4.5%
10	5	30	771	742	3.7%

Table 2. Performance prediction using model: n=550

4.2 Scalability

If the simulation is distributed on M machines, the expected speed up factor will conventionally be equal to M. For small size simulations, it is not possible to achieve this figure due to the presence of inter-node communications and thread synchronizations which dominate the simulation time. With large scale simulations, however, it is possible to achieve scalability greater than M. The performance model also demonstrates this fact clearly. The speedup factor, the improvement in execution time by running the simulation on several Grid nodes compared to that on a single machine is given by the ratio of two polynomials in N. For large N, the quadratic term will be dominant, and the speed up factor asymptotically approaches the ratio $a_2(M)/a_{21}$ where $a_2(M)$ is the coefficient of the quadratic term in (4) and a_{21} is the corresponding coefficient for a single machine case. These are further illustrated in the plots shown in Figure 2.

Fine-grained applications take longer execution time because for a fixed work load, fine graining an application increases overhead as a result of thread context switching needed to maintain synchronization. If a thread completes a loop before its assigned quantum is finished, it should be forced to yield control. However, depending on the scheduling policy of the JVM implementation, it can always be on top of the ready queue. From the scheduler view point, this thread did not get a fair share of the CPU time and should be run before other threads, while from the application point of view, it has up-to-date temporal status and should wait until all threads do likewise. Coarse grained applications are more efficient, with less frequent involuntary yielding by threads.

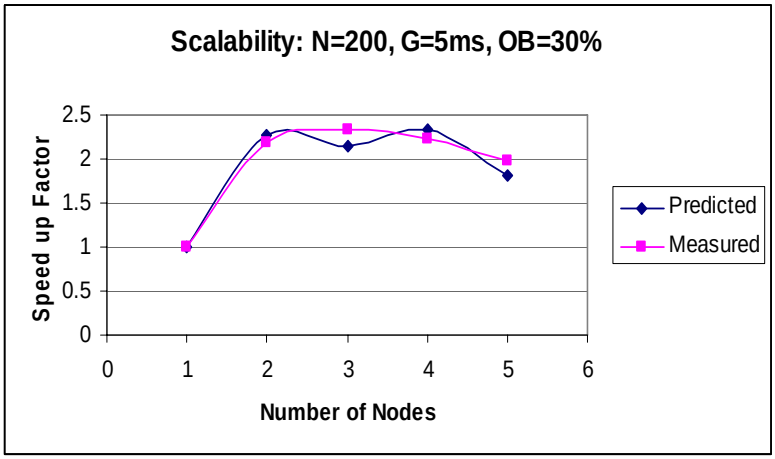

Fig.1a. Small simulations may not be always scalable

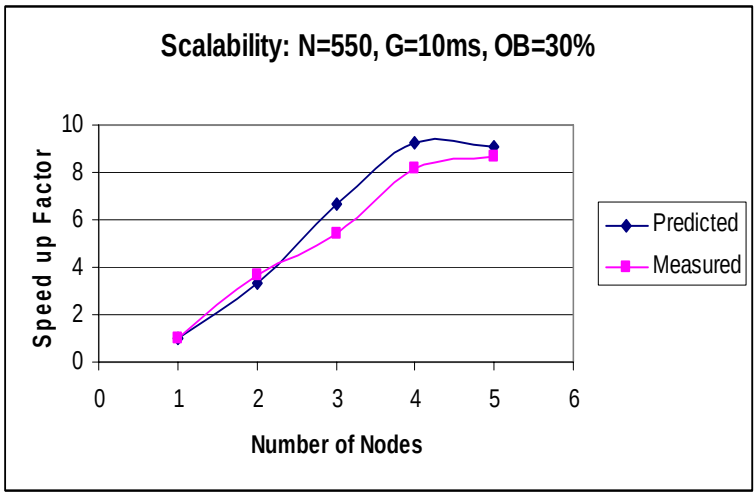

Fig. 1b. Better scalability for large simulations.

As can be seen in Fig. 2b, the determination of M for maximum performance is not always trivial. For fine-grained applications, running the simulation over several nodes may not always yield the desired speed up. In fact, it may have the opposite effect, (depending on the number of agents). The performance predictor can determine the arrangement (such as the number of nodes, the number of agents per node, etc) for optimized operation.

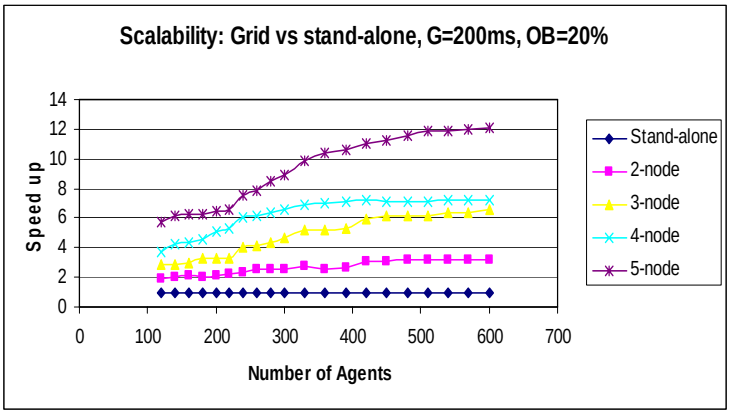

Fig.2a. Coarse-grained simulations

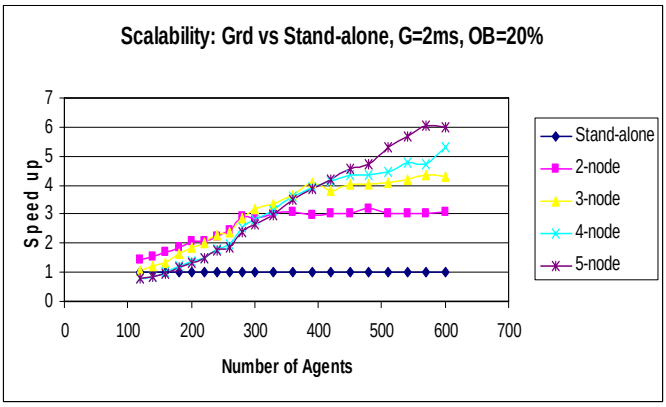

Fig. 2b Fine grained simulations

4.3 Prediction with Variable application Granualirty:

The validity of the predictor was verified using a workload having the feature of real simulation applications. The granularity of the new workload is expressed statistically. It has a uniform distribution with a maximum range of deviation (d%) from a mean granularity value (μ). The execution times for different deviation ranges were compared against the predicted performance for the mean granularity. Figure 3 shows that even with 60% deviation range, the prediction is fairly good. The accuracy of the prediction improves with N.

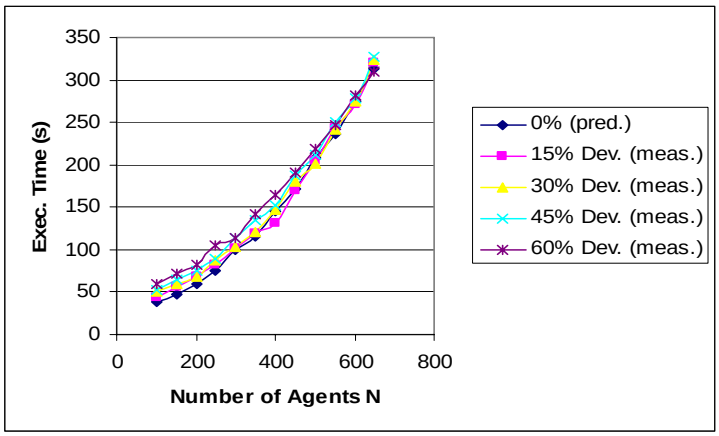

Fig.3. Effect of granularity variation on prediction. (OB=80%, 5-node) for different deviation vales (μ = 20ms)

4.4 System Utilization:

The measured CPU utilization data shows that for small scale simulations, the system is idle for a considerable part of the time. This idle time is a measure of performance loss associated with the communication latency. Although utilization is better for large simulations, a breakdown of the time into *user-thread* and *system-thread* usages shows that a significant time is spent on system threads too.

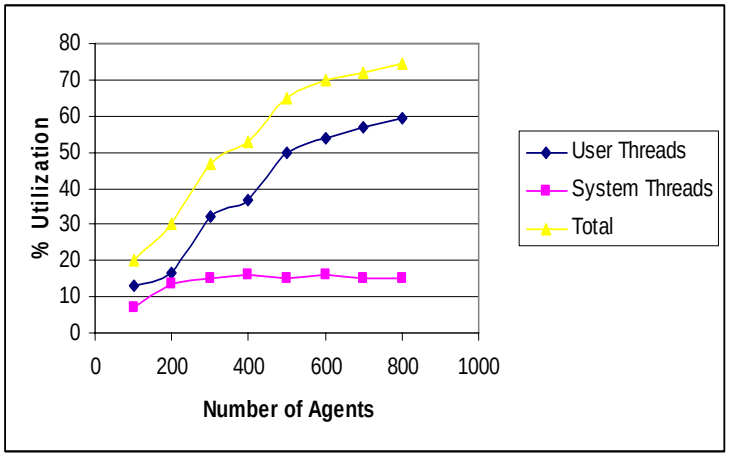

Fig.4. CPU Utilization for OB=80%, 5-node

5. Discussion

The overall performance characteristics observable from the response time and speed up plots match with the predicted trends. The quadratic behaviour of the response time when the workload size or the number of threads increases can be attributed to several factors, among which the following are more likely [14]:

1. The increase in number of synchronized objects introduces delays associated with locks.
2. The JVM decision on code optimization, garbage collection, scheduling policy, etc., is not clearly understood.
3. Operating system behaviour that cannot be adequately captured to be included into the performance model and the predictor.

The contribution of each factor is not exactly known, but the combined effect is clearly visible in the unpredictability of the threads execution sequence, a major problem causing excess overheads during time-step synchronization.

Since the Grid is a dynamic environment, the list of factors contributing to unpredictability is even longer and it would be necessary to build dynamic performance models and handle load monitoring and management throughout the execution.

Limitations of the Study

- The performance model built is not general and usable over a wider range. This requires a larger experiment with rigorous modeling and analysis techniques.
- The experiment is conducted under a controlled environment and important factors such as heterogeneity of resources, communication bottlenecks, load conditions, etc., are not accounted for.
- The accuracy of the prediction model for variable granularity cannot be generalized since granularity may have different stochastic distribution models.

The findings of the experiment will thus need to be further improved to address the above limitations.

6. Conclusions and Future Work

The experiment shows how performance prediction for MABS application models can be built by incorporating the prominent features of MABS. Since the performance model is built for a simplified workload, additional effort is needed to achieve a more comprehensive model. As it is now, however, it can provide a general guidance to MABS application developers and simulation modelers on how to address design issues related to performance and application modeling.

To make the best use of performance predictor for MABS applications, it can be placed as a middleware layered between the MABS application and the standard Grid services. In practical deployment scenarios, application parameters may not be well characterized or are not known in advance. Thus, they should be determined on the fly and the middleware can prove a useful tool for this. This is particularly essential if performance prediction for dynamic load management is the main purpose.

Another issue is the relation between high level simulation modeling and the architecture of the MABS application to be deployed. Coarse grained architectures generally provide better performance and it is important that simulation modelers can design their application with this factor in mind. Some MABS application models allow bundling of several agents into one thread instead of deploying them as separate threads. The performance trade-off achievable by using this approach will be seen in future experiments.

7. References

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

[2] Davidsson, P. et al. “Applications of multi-agent based simulations,” *Seventh Workshop on Multi-Agent-Based Simulation* (2006), Future University-Hakodate, Japan.

[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. “Smooth scaling ahead: Progressive MAS simulation from single PCs to Grids,” *Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation* (2004) New York

[5] Ferreira L. et. al. “The IBM Red Book. Introduction to Grid Computing with Globus”, (2003).

[6] Sotomajor, B. “The Globus toolkit 4 Programmer’s Tutorial”, (2005).

[7] Helsinger, A. et al. “Tools and techniques for performance measurement of large distributed multi-agent systems,” *AAMAS’03* (2003) Australia

[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] Tirado-Ramos, A., Groen D., Sloot, P. “On-line Application Performance Monitoring of Blood Flow Simulation in Computational Grid Architectures,” *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*, (2005)

[10] Iskra, K.A., Albada, G.D., Sloot, P.M.A. ”Towards Grid-Aware Time Warp”, *Proceedingsof the 18th Workshop on Parallel and Distributed Simulation, (*2004)

[11] Puppin D., Tonellotto N., and Laforenza D. “Using Web Services to Run Distributed Numerical Applications”, in LNCS vol. 3241 D.Krazmuller Springer-verlag Berlin Heidelberg (2004) pp. 207-214.

[12] Timm I.J., Pawlaszczyk D., “Large Scale Multiagent Simulation on the Grid,” *Proceedings of 5th IEEE International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Washington, DC, USA, (2005).

[13] Barnett J., “The behaviour of Java threads under Linux NPTL” , (2003).

[14] Tichy, W.F. “Should computer scientists experiment more?,” *IEEE Computer*, USA. Vol. 31 No. 5 (1998), pp.32-40

[15] 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

[16] Badia, Rosa M., "Performance Prediction in a Grid Environment," *1st European Across Grids Conference*, Santiago de Compostela July 2003.

[17] Jarvis, S.A. et. al. "performance-responsive middleware for grid computing", *Proceedings of UK e-Science All Hands Meeting, (2003) Nottingham, UK.*

[18] Collis J., Ndmu D., Buskirk C., "The Zeus Agent Building Toolkit", *(2000).*

SIX

A Grid Based Simulation Environment for Mobile Distributed Applications

Dawit Mengistu, Paul Davidsson, Lars Lundberg

Department of Systems & Software Engineering,
Blekinge Institute of Technology, Ronneby, Sweden
{dawit.mengistu, paul.davidsson, lars.lundberg} @bth.se

Abstract

Using simulation environments to support mobile distributed application development has many benefits. Since mobile devices can be modeled as autonomous intelligent agents, the multi agent based simulation (MABS) approach is useful for such applications. In this paper, we propose a MABS platform on the computational Grid for mobile distributed computing. A prototype of the proposed platform is implemented and its performance characteristics are studied. The findings of the experiment show that a Grid based MABS platform can provide a scalable simulation environment for mobile distributed applications.

1. Introduction

Innovations in wireless communications and embedded systems have brought new areas of mobile distributed computing (MDC) applications into the scene. The advances in these technologies have led to a major breakthrough in ubiquitous computing.

The mobility of users together with their computing devices and the special characteristics of mobile environments, such as highly variable connectivity, disconnection, location dependency, and energy and resource sensitivity, and the diversity and flexibility introduced by mobile systems bring new challenges for research in distributed computing. [2]

To cope with the ever increasing demand for new applications in MDC and to improve the performance of the existing ones, there is an eminent need

for a simulation environment. Simulation tools are essential to understand the operation of the systems, gain deeper insight into current problems, evaluate scenarios, and propose new architectures. However, due to the characteristics of MDC, traditional macro simulation techniques may not be effective.

Parunak [19] lists the following characteristics for an ideal application of agent technology:

- Modular, in the sense that each entity has a well-defined set of state variables that is distinct from those of its environment and that the interface to the environment can be clearly identified.
- Decentralized, in the sense that the application can be decomposed into stand-alone software processes capable of performing useful tasks without continuous direction from other software process.
- Changeable, in the sense that the structure of the application may change quickly and frequently.
- Ill-structured, in the sense that all information about the application is not available when the system is being designed.
- Complex, in the sense that the system exhibits a large number of different behaviours which may interact in sophisticated ways.

Clearly, most MDC applications share these characteristics. The autonomy and intelligence of components in mobile distributed systems is another incentive to consider agent based modeling and simulation techniques for MDC applications.

However, MDC applications consist of a large number of entities which would be very difficult or impossible to simulate within an acceptable span of time without substantial computing resources. One way to alleviate this problem is to exploit the computational power offered by the Grid.

The Grid offers a robust distributed computing platform for agent based applications since it allows the agents to discover, acquire, federate and manage the capabilities necessary to execute their decisions [8]. It enables the creation of problem solving environments such as simulation platforms through which end users run complex applications that usually require collaboration and interaction of a large amount of distributed heterogeneous resources [1].

In this paper, we present a high level architecture of a dynamic simulation environment for MDC deployed on the Grid as a multi-agent based simulation (MABS) application. A major challenge is ensuring scalability and

performance of MABS applications in a distributed environment such as the Grid. The inherent social behaviour of agents requires that they interact or communicate with each other intensively. This inter-action involves frequent data exchange between agents running on different Grid nodes. We implemented a prototype of a proposed model and studied issues related to performance and scalability.

The rest of the paper is organized as follows: a brief background, motivations for this study and related work will be presented, followed by discussion of the multi-agent modeling approach. We will then present the proposed simulation architecture followed by a description of the experimental prototype and the preliminary results. A discussion of the important findings of this work and some pointers to future work concludes the paper.

2. Background and Motivations

MDC is a ubiquitous computing environment with a complex application architecture where several devices and users are wireless and mobile. Due to frequent changes in location and resource availability, applications in mobile environments should be able to adapt to fluctuations in resource availability and changes in the location of nodes.

Mobile devices that communicate through wireless channels typically have fewer resources and less processing power, and thus cannot perform resource-intensive tasks. Energy constraints imposed on mobile devices by small batteries is another motivation to move the computational load elsewhere. By redistributing the computational load on demand, the users of mobile devices would be able to solve far more complex and resource-demanding problems that they would be unable to solve using their individual devices on a standalone basis. [6]

MDC has several general and context specific challenges that need to be studied using prototype models or through simulation methods:

- Requirements for deploying new systems involving cooperative problem solving applications, to enable mobile devices with limited resources to solve problems that they cannot handle individually.
- Reviewing/refining existing architectures and implementations to cope with upcoming needs of resources to improve performance.

- Ensuring reliable and optimal use of MDC infrastructure such as network bandwidth and resources of individual devices.
- Configuring the components dynamically to adapt to changes in the environment, spontaneous inter-operation of all hardware and software, a basic necessity of ubiquitous computing systems.

A simulation environment may not be effective if it does not accommodate the salient features of the system to be studied. An agent based simulation approach thus seems promising for capturing the relevant features of MDC and conducting different studies.

3. Related Work

A similar approach is proposed in the Caribbean system [14], a massively multi-agent platform for use in large scale navigation system of humans carrying mobile terminals. The system takes into account the effect of the behaviour of social systems that includes humans. It is recognized that running real world experiments is very hard for such systems and instead, multi-agent simulations are used where each person is modeled as an agent. The limitation of this simulation tool, however, is that it is developed as a stand alone application and does not scale up to a distributed execution environment.

Shibuya [15] proposes a framework of multi-agent based modeling and simulation in a ubiquitous environment. It assists computational modeling and quantification of social behaviour in mobile systems, contextualizing location based services: applications that use spatio-temporal information and a geospatial information system. However, implementation details such as deployment platform and infrastructural issues, however, are not described.

There are a number of relevant contributions that do not consider simulation per se, but shows how agent approached can be used for MDC applications. For instance, Gunderson and Petersen [16] propose a multi-agent based problem solving approach in mobile environments. It is based on the EU AmbiSense project (IST-2001-34244). It mainly focuses on dynamic planning issues of applications running on mobile devices such as task decomposition, match making for task allocation and result collection. The work is at a prototype stage and no empirical validations have come out yet.

A development environment called MADE [17] was developed for intelligent applications on mobile devices to simplify the development of applications using the agent approach. It is, however, limited to writing a prototype standalone version of the application, and does not consider issues that emerge with the interaction and interoperation of the devices.

An application that uses the JADE multi-agent platform on mobile devices was developed by Moreno et al. [18]. It implements personal agents on PDA devices to solve a particular problem in a specific setting. However, it cannot be used as a simulation environment as is now.

Although efforts have been made to employ the multi-agent paradigm for MDC applications, most of them have addressed controlling and coordination issues. The work involving simulation concerns either single machine simulations or are only conceptual proposals. We argue that the potential multi-agent approaches have not yet been fully exploited due to lack of a generic agent based platform to distributed simulations. The main purpose of our work is to build an environment that facilitates large-scale agent-based simulations for MDC using the Grid infrastructure.

4. Multi-agent Modeling

Although a multi-agent system is characterized by distributed behaviour, a core property of the agents is that they achieve their objectives through interaction, negotiation and cooperation. A key to a multi-agent system is the sociability of the agents and the presence of some kind of logical organization of the agents based on their functions. This organization is in most cases role oriented and defines the interaction links that can exist between agents.

As explained earlier, MDC components tend to be built out of the composition of mobile, autonomous devices and systems. A mobile distributed system may thus be considered as a multi-agent system where devices, users, infrastructure, etc., are regarded as autonomous agents capable of performing their own tasks, sharing resources available to them and commu-nicating with other agents.

MABS can assist MDC applications because: [11]

- It supports modeling and implementation of proactive behaviour, which is important when simulating the human users who take initiatives and act without taking external stimuli

- It is possible to add or remove agents during simulation without interruption
- An agent can be replaced with the corresponding simulated (physical) entity while the simulation is running. This shows that the MABS can preserve the structural details of the simulated system.
- It supports distributed computation in a very natural way. Since each agent is typically implemented as a separate piece of software running in its own thread of control, it is possible to run different agents on different machines.
- The mobility of devices can be captured using the agent model.
- It is very difficult to characterize and validate the system using explicit mathematical models, even if it is possible, it is not easy to find analytical solutions as the model will be too complicated

The simulation provides the user with interesting patterns and results that may invite further analysis. Emergent behaviour from the interactions between the agents at a micro level is not usually observed with other modeling approaches. Therefore, MABS, an approach replacing traditional simulation techniques in the analysis of complex systems, seems very useful to study mobile distributed systems.

We argue that the Grid is the right environment for running MDC simulations because it provides the much needed computational resources to execute the simulations as MABS applications.

5. The Simulation Environment

As building agent applications from the scratch is time consuming, the use of platforms as an environment for deploying MABS applications has become popular. A MABS platform provides the physical environment in which agents are deployed. It typically consists of an agent service, a directory service, and a message transport service.

Furthermore, to implement the components of the platform on the Grid as a distributed application, a middleware is required. Figure 1 shows a schematic of the architectural model of the proposed simulation platform. An experimental prototype of the model is developed to study important implementation issues.

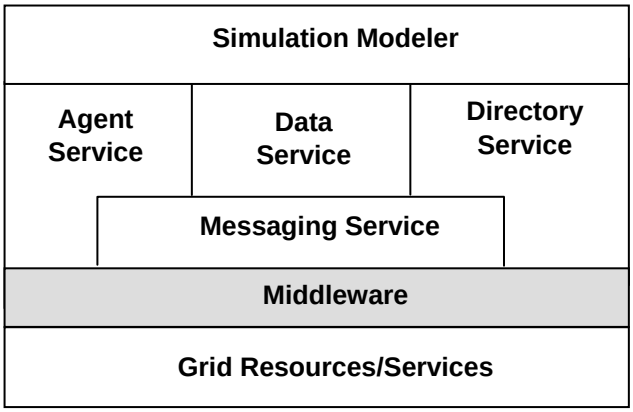

Figure 1. Proposed Architecture

The Directory Service maintains information about the agents such as naming, locating, organizing, etc. Upon launching, the agents automatically register themselves in the directory service with a unique identifier. It has a hierarchical architecture distributed over the Grid nodes involved in simulation, 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 transformed into a knowledgebase which the agents regularly use to update their beliefs about the environment.

The *Agent Service* is used to handle system level functionalities such as the launching and termination of agents, their registration and deregistration with the directory service.

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

The *Simulation Modeler* allows users to build models of the entities to be simulated, such as mobile devices, users, base stations, etc. It provides the environment to write the program codes corresponding to the tasks of the simulated entities.

The *Middleware* is responsible for interfacing the platform functionalities with the Grid resources. It works with all services of the platform for:

- distributing the task to available Grid nodes,
- reallocation of agents and migration of tasks
- monitoring the performance of the application etc.

Grid schedulers can employ the support of the middleware for load balancing, task reallocation and migration. The middleware monitors the application's communication and computational characteristics to decide on the optimal allocation strategy.

6. Experiment and Results

A prototype of the proposed MABS platform is implemented on an experimental Grid testbed running the de facto standard Grid middleware, the Globus Toolkit version 4. At this stage, only generic agents with basic functionalities are implemented to study the general performance characteristics. Because MABS applications are characterized by high communication-to-computation ratio, their performance on the Grid highly depends on the communication behaviour of the agents. The focus of the current experiment will therefore be on this aspect.

The architecture of the application is quite similar to familiar Grid applications. It is run as a parallel web service application launched from a master (client) node where the workload is partitioned into balanced loads and distributed to worker nodes on the Grid.

The Grid nodes are Linux machines having PIII 1000MHZ processors with 512MB RAM. The Globus version is GT4 running on Java 1.4.2 and Ant 1.5. The machines are connected via a 100Mbps switch to the Internet. Some components of the Globus toolkit such as GFTP and security features, although installed, are not considered since their presence bears no relevance to the performance experiment. Each node hosts an identical copy of the MABS application as a Web Service in its container. Another machine with identical specification is used as a client node to launch the application and manage the basic platform services shown in the architectural model of Fig. 1.

As said earlier, the simulation space is evenly divided into the number of Grid nodes participating in the simulation, with each node hosting equal

number of agents. The agents are instantiated as user threads in the Grid web service when the client node launches the application. As the simulation size increases, the number of agents, thus the number of threads to be launched also increases. However, because large-scale multi-threading undermines performance significantly, it is necessary to work the best deployment method. The experiment is designed in such a way that during one simulation cycle, each agent first executes a fixed task and then sends a message to another agent. This cycle of computation-communication is executed multiple times (equal to the number of time steps in the simulation). The following parameters are used as control variables in the study:

- number of agents (shown as number of threads)
- outbound communication rate (OB in %), the ratio of the number of messages to agents residing in 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 in each time step (before sending the message)
- number of Grid nodes

To relieve the network from congestion, messages destined to a node are aggregated by the Messaging Service. Similarly, messages leaving a node are collected together and dispatched once every time step. The messaging agent usually delivers or receives several messages when it gets control of the CPU.

An important feature of the application, typical of MABS implemented as time-driven simulation, is the need for synchronization among the threads to maintain coherence with respect to the simulated time. All threads should advance execution at a synchronized pace of discrete steps to imitate real time. Threads which have completed a one time-step task remain in a waiting state by yielding CPU control voluntarily, until all threads get to the same level.

If an agent sends out a message and does not receive a reply immediately (which is the case in most situations), it voluntarily yields control of the CPU and stays in the wait state until a reply comes. It will be forced to return to the wait state, even if it rightfully gets CPU time again to be in the run state before the reply is received. In such cases, depending on the granularity of the application, the waiting time may be higher than the useful computation time. This waiting can be even longer if the reply is to come from another node, that is, if the messaging is outbound.

In coarse-grained simulations, the time a thread spends in the regular wait state, due to preemption imposed by system scheduler, (without messaging wait) is relatively long. As a result, there is a higher chance for the expected reply to arrive before the thread goes into the run state again.

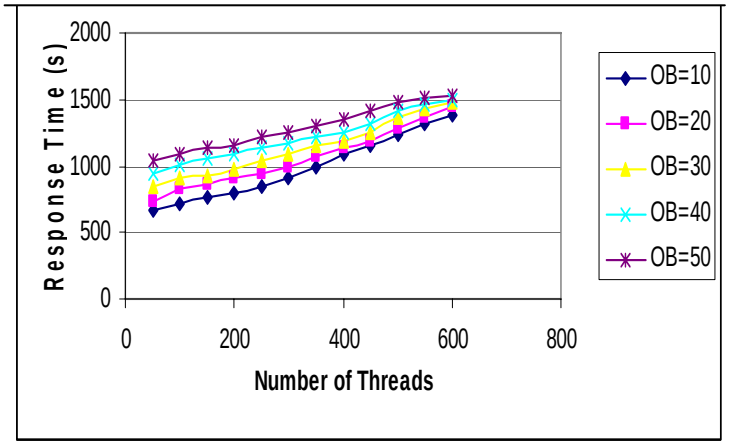

Figure 2. Effect of outbound communication on a fine-grained workload, 5-node Grid.

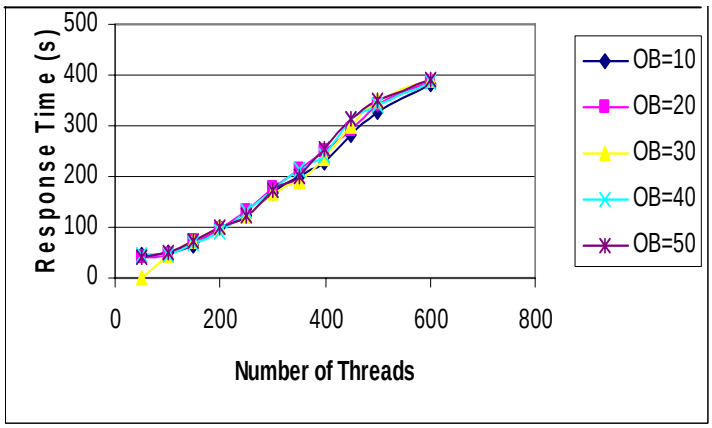

Figure 3. Effect of outbound communication on a coarse-grained workload, 5-node Grid.

Figures 2 and 3 compare the effect of outbound communication on fine and coarse-grained simulations. The granularity is the amount of work a thread executes in one time step (before it performs the communication

operation), measured as the time it takes to finish this work. Granularity sizes of 2ms and 200ms respectively are used to represent fine and coarse-grained tasks through out this paper for demonstration purposes. The response time is used as a measure of the application's performance in a particular configuration.

Figure 2 shows that if the granularity of the workload is small, it will not be possible to achieve a satisfactory scalability since threads spend most of the time in waiting state (yielding CPU. The problem gets even worse when the outbound communication rate is higher. It is also seen that for large simulations the gap between high and low outbound communication rates gets narrower. The reason is that, as the level of multi-threading increases, system overhead, independent of communication behaviour, also increases.

In the coarse-grained workload like the one shown in Figure 3, the variation in execution time between the lowest and highest outbound communication for the same simulation is not remarkably high. It can therefore be said that the influence of outbound communication on performance is less significant if the granularity of the simulation is coarse. This is due to the fact that the waiting time of threads in coarse grained tasks is mainly that of preemption by the scheduler. For fine grained and communication intensive simulation workloads, how-ever, deploying the application on several nodes is a liability as Figure 4 illustrates.

Another contribution to performance degradation inherent to MABS applications is the need for time-step synchronization explained earlier. If threads are required to yield voluntarily before their allotted quantum expires, a lot more context switching overhead occurs, with a severe effect on performance.

The synchronization is implemented at two levels: local and global. At the local level, all threads within a node should synchronize. At the global level, all nodes send synchronization signals to the master node when a local level synchronization is achieved. Thus, since time synchronization is implemented centrally, all agents in the simulation should advance in harmony. Therefore, a potentially ready-to-run thread on one node may be stalled for a long time, waiting for all threads, including those running on different nodes. The waiting time here also can be much higher than the useful computation time, as a result of which the performance becomes poor.

The loss in application scalability is more severe in fine-grained applications. Coarsening the application granularity improves scalability since

the loss due to context switching overhead and waiting time attributed to the two factors (outbound communications and time-step synchronization) will be dominated by the regular thread wait time imposed by the scheduler.

The figures 4 and 5 illustrate the impact of application granularity on scalability. The curves show the speedup gain in fine and coarse-grained simulations respectively, taking a stand-alone implementation as a reference.

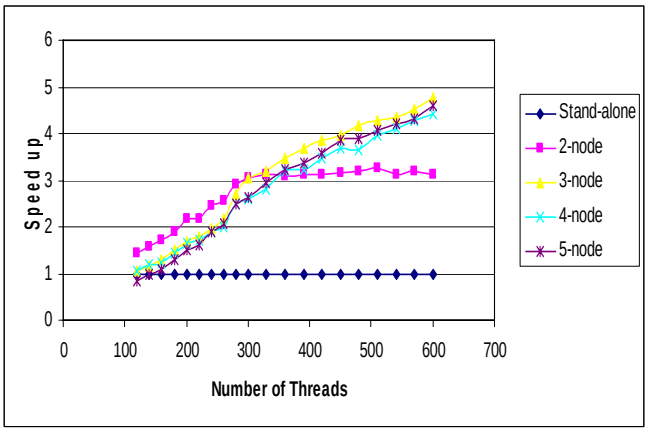

Figure 4. Effect of granularity: speed up for a fine-grained workload.

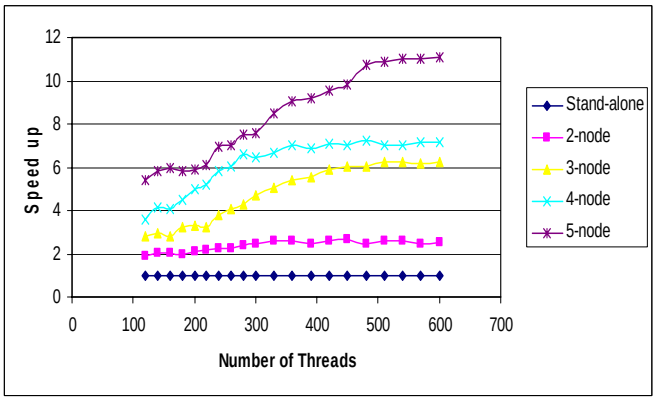

Figure 5. Effect of granularity: speed up for a coarse-grained workload.

The above performance characteristics and scalability can also be examined in a different way as shown in the following figures. For small-scale simulations, the Grid offers poor scalability. Particularly in fine-

granular simulations, the speedup is low and in some cases it can be faster to run the simulation on a single stand-alone computer. In these figures too, the speedup is computed taking the stand-alone implementation as a reference.

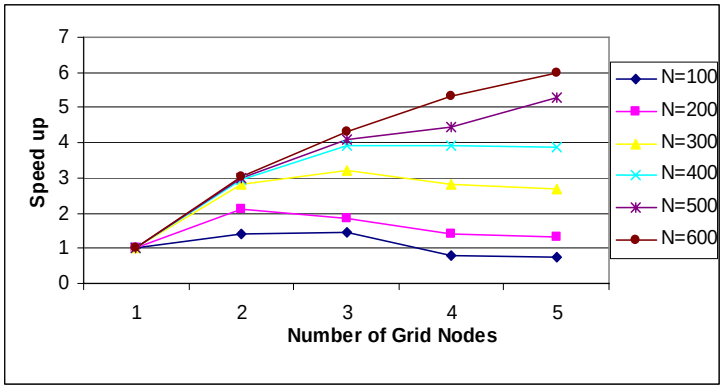

Figure 6. Application scalability of a fine-grained workload, N= Number of threads

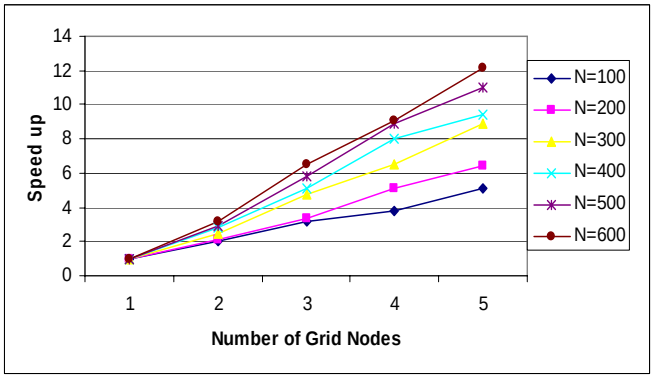

Figure 7. Application scalability of a coarse-grained workload, N= Number of threads

The overall performance characteristics as can be seen from the response time plots match with the expected trends although they get more non linear with increase in the simulation size. Possible reasons are:

1. The presence of synchronized methods and creation of dynamic objects during run time makes the JVM very inefficient, particularly when the number of simulation threads is large.

2. The complexity of JVM and Operating System decisions on scheduling, garbage collection, code optimization, etc.
3. The scheduling policy implemented by the JVM affects the sequence of execution of threads, in turn affecting the predictability of messaging services, causing longer wait times on those threads which expect reply messages.

7. Conclusion and Future Work

The prototype developed has given important information on the features of MABS that should be studied during simulation design and demonstrated that the Grid can support scalable large-scale distributed agent based simulations. Mobile distributed systems, which constitute a large number of interacting entities with different behaviours can therefore be simulated using the MABS approach on the Grid.

In order to focus our study on major performance issues of the platform, the prototype did not include the details and complexities involved in real applications for the time being. In the next work, we shall use more realistic simulations models incorporating the most salient features.

The proposed middleware can be used to improve the performance of fine grained MDC simulations by reorganizing the application, such that several agents are bundled into a single thread. If agents having more frequent communications are bundled together in to a common execution thread, the gain in performance would be high since most communication becomes internal within the thread itself.

One way to reduce the communication overhead is to minimize outbound messaging. MABS, however, have inherent limitations to employ this technique directly. A feature can be added into the middleware so that it can, by studying the messaging patterns, redeploy the agents in such a way that outbound communication is reduced. This involves online monitoring of the application, dynamic load balancing and task migration. Such features make the simulation environment more robust, in addition to improving application performance.

Several cooperative distributed problem solving applications in the mobile environment, such as traffic congestion management simulations, can be

simulated using this architecture. Another area of usage is infrastructure and network studies in wireless devices.

8. References

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

[2] Issary, V. et.al. "Software Architecture for Mobile Distributed Computing". *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture"*. (2004)

[3] Abdu, H. A Framework for Reconfiguring Mobile Application. "*Proceedings of the 25th IEEE Conference on Distributed Computing Systems Workshops*". (2005)

[4] Helisinger, A. et.al. "Tools and Techniques for Performance Measurement of Large Distributed Multiagent Systems". *Proceedings of the second international joint conference on Autonomous agents and multiagent systems* ACM Press New York, USA, (2003), pp. 843-50.

[5] Barbeau, M. "Mobile, distributed, and pervasive computing" *Handbook of wireless networks and mobile computing archive.* John Wiley & Sons, Inc. New York, USA, (2002), pp.581-600.

[6] Kurkovsky, S. and Bhagyavati "Wireless Grid Enables Ubiquitous Computing", *Proceedings of The 16th International Conference on Parallel and Distributed Computing Systems(PDCS-2003)*, Reno, NV, (2003)

[7] Kurkovsky S., et. al. "Modeling a Grid-Based Problem Solving Environment for Mobile Devices", *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*, Las Vegas, Nevada, (2004)

[8] Foster, I., Jennings, N.R., Kesselman, C. "Brain meets brawn: Why Grids and agents need each other" *Proc. 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (2004) New York*

[9] Roy, N. et. al. "Enhancing Availability of Grid Computational Services to Ubiquitous Computing Applications" *Proceedings of the 19th IEEE*

International Parallel and Distributed Processing Symposium, IEEE Computer Society Washington DC, USA, (2005).

[10] Modahl, M. et. al. "Towards a Standard Ubiquitous Computing Framework." *2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC'04)*. Toronto, Canada, (2004).

[11] Davidsson, P. "Multiagent Based Simulation: Beyond Social Simulation". *Springer Verlag LNCS series* Vol 1979, (2000)

[12] Maria, A. "Introduction to Modeling and Simulation." *Proceedings of the 29th conference on Winter Simulation"*Atlanta, Georgia, USA, (1997). pp.7-13

[13] Kurkovsky S., Bhagyavati, and Ray, A., "Modeling a Grid Based Problem-Solving Environment for Mobile Devices." *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)* Vol. 2. IEEE Computer Society Washington, DC, USA (2004) pp. 135-136.

[14] Nakajima, Y. et.al. "Caribbean/Q: A Massively Multi-Agent Platform with Scenario Description Language" *Second International Conference on Semantics, Knowledge, and Grid, SKG '06* (2006).

[15] 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.

[16] Gunderson, O.E., and Petersen A..K. "Multiagent Based Problem-solving in a Mobile Environment" *Norsk Informati-kkonferanse* Bergen Norway, (2005)

[17] Hall, L. et.al. "A Development Environment for Intelli-gent Applications on Mobile Devices" *Expert Systems with Applications Vol. 27, No. 3, Elsevier Ltd. 2004* pp. 481-92

[18] Moreno, A., Valls, A., and Viejo, A. "Using JADE-LEAP to implement agents in mobile devices" Available on- line: http://jade.tilab.com/papers/-EXP/02Moreno.pdf: visited July 2006.

[19] Parunak, H.V.D. "Industrial and Practical Applications of DAI", *Multiagent Systems,(ed.) G.Weiss. MIT Press. (*1999).

ABSTRACT

The computational Grid has become an important infrastructure as an execution environment for scientific applications that require large amount of computing resources. Applications which would otherwise be unmanageable or take a prohibitively longer execution time under previous computing paradigms can now be executed efficiently on the Grid within a reasonable time.

Multi-agent based simulation (MABS) is a methodology used to study and understand the dynamics of real world phenomena in domains involving interaction and/or cooperative problem solving where the participants are characterized by entities having autonomous and social behaviour. For certain domains the size of the simulation is extremely large, intractable without employing adequate computing resources such as the Grid.

Although the Grid has come with immense opportunities to resource demanding applications such as MABS, it has also brought with it a number of challenges related to performance. Performance problems may have their origins either on the side of the computing infrastructure or the application itself, or both.

This thesis aims at improving the performance of MABS applications by overcoming problems inherent to the behaviour of MABS applications. It also studies the extent to which the MABS technologies have been exploited in the field of simulation and find ways to adapt existing technologies for the Grid. It investigates performance monitoring and prediction systems in the Grid environment and their implementation for MABS application with the purpose of identifying application related performance problems and their solutions.

Our research shows that large-scale MABS applications have not been implemented despite the fact that many problem domains that cannot be studied properly with only partial simulation. We assume that this is due to the lack of appropriate tools such as MABS platforms for the Grid. Another important finding of this work is the improvement of application performance through the use of MABS specific middleware.

ISSN 1650-2140

2007:06

ISBN 978-91-7295-118-1