International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

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

Dawit Mengistu, Lars Lundberg, and Paul Davidsson

***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 to estimate the performance characteristics of real world simulation applications.

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

I. Introduction

THE computational Grid has emerged in the past years as a key technology to support the execution of 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.

II. Multi-Agent Based Simulation on the Grid

A. 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.

B. MABS Features

A detailed list of the features characterizing MABS applications is discussed in [2]. As some of these features

Manuscript received March 25, 2007.

The authors are with Blekinge Institute of Technology, Soft Center, 372 25, Ronneby Sweden (e-mail: dawit.mengistu@bth.se, lars.lundberg@bth.se, paul.davidsson@bth.se).

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 with in 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 a lot of idle time for the communicating agents if the next course of action the agents 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.

C. 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.

III. Methodology

A. 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 clearly separated 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.

B. 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.

C. 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 is reduced to that of the application's behaviour, the effect of communication and task granularity.

The envisaged performance model will have the following input parameters:

- 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:

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

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

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 $a_0, a_1, \ldots, a_m$ 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.

D. 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. Since the applications run in both environments are identical, we only discuss the Grid version from now on.

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.

IV. Results

A. 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 I and II. The chosen values of the inputs are selected from the model's valid ranges.

TABLE I
PERFORMANCE PREDICTION USING MODEL: N=180

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 II
PERFORMANCE PREDICTION USING MODEL: N=550

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%

B. 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 Fig. 2.

Fine-grained applications take longer execution time because for a fixed work load, fine graining an application increases overhead and thread context switching due to yields, 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 may 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.

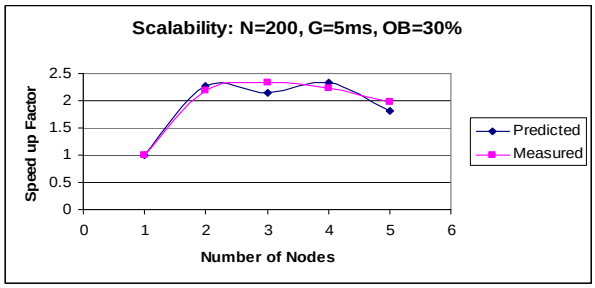

Fig. 1(a) Small simulations may not be always scalable

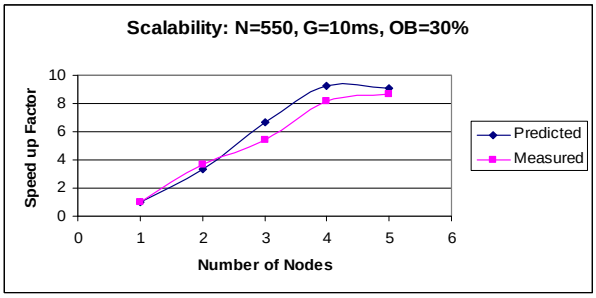

Fig. 1(b) Better scalability for large simulations

As can be seen in Fig. 2(b), 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.

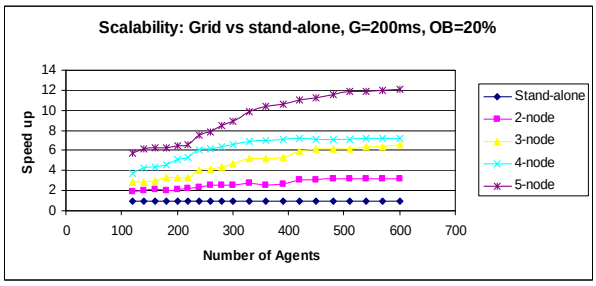

Fig. 2(a) Coarse-grained simulations

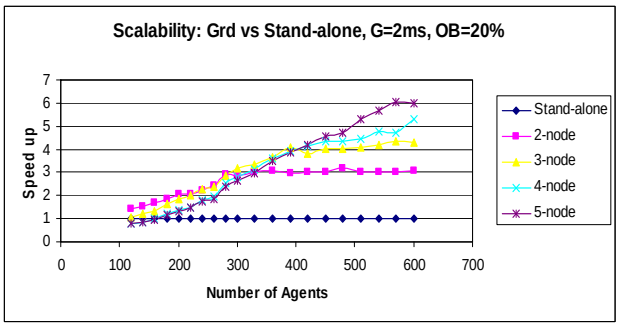

Fig. 2b Fine grained simulations

C. *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 given in terms of 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. Fig. 3 shows that even with 60% deviation range, the prediction is fairly good. The accuracy of the prediction improves with N.

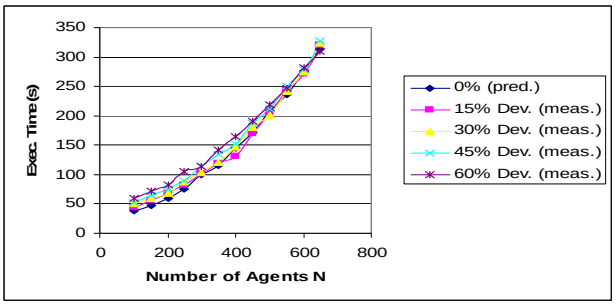

Fig. 3 Effect of granularity variation on prediction (OB=80%, 5-node) for different deviation vales (μ = 20ms)

D. *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.

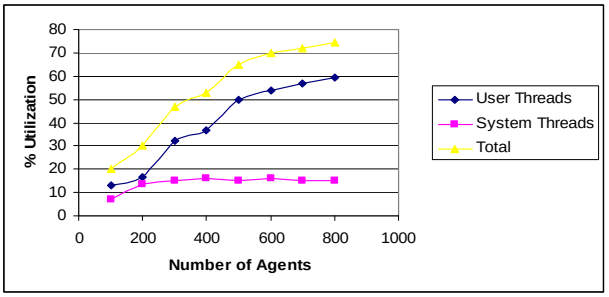

Fig. 4 CPU Utilization for OB=80%, 5-node

V. 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 clearly 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.

VI. Conclusion 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.

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).*