Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

PERFORMANCE EVALUATION OF A CMB PROTOCOL

Célia L. O. Kawabata

Centro Universitário Central Paulista
Rua Miguel Petroni, 5111
São Carlos, SP 13563-470, BRAZIL

Regina H. C. Santana
Marcos J.Santana
Sarita M. Bruschi

ICMC, Universidade de São Paulo
Av. Trabalhador São-Carlense 400
São Carlos, SP 13560-970, BRAZIL

Kalinka R. L. J. Castelo Branco

Centro Universitário Eurípides
de Marília
Av Hygino Muzzi Filho 529
Marília, SP 17525-901, BRAZIL

ABSTRACT

This paper presents the performance evaluation of a CMB (Chandy-Misra-Bryant) protocol from the perspective of execution time. The performance of each logical process in simulation is measured. Our evaluation shows that logical processes can have different behaviors and different protocols can be used simultaneously in simulations. While some logical processes may perform well using conservative protocols, others can use optimistic protocols because otherwise most of the time these processes would be blocked unnecessarily. In order to analyze the behavior of the simulations some models were simulated using a CMB implementation called ParSMPLX. These models showed that each logical process of a simulation has a different behavior that makes it more suitable for a specific protocol, increasing the performance.

1 INTRODUCTION

The simulation technique is a powerful tool to evaluate performance of computer systems, however it can consume a lot of time and computational resources. Even with fast processors, simulations can take many hours to complete. In order to reduce this time, simulations can be executed in parallel machines or distributed systems (Dongarra et al. 2002, Kumar et al. 2004, Fujimoto 2003).

One parallel simulation approach is to decompose the simulation model into logical processes and simulate each one in a different processor. This approach can reduce the simulation time for some applications, mainly those which are easy to become parallel and have large computational granularity.

Research has been done in distributed simulation technique and most of these studies focus on two well known protocols: conservative and optimistic (Bruschi et al. 2004, Xu and Chung 2004, Bauer et al. 2005, Bononi et al. 2005, Curry et al. 2005, Lee, Luu and Konangi 2005).

Events in the conservative approach are executed only when causality errors can be avoided, i.e., when there is not an event with smaller timestamp than the first event in the event queue.The optimistic protocol, on the other hand, simulates all the events without being concerned about causality errors. If an error occurs, the simulation returns to a safe state (using rollbacks) and the simulation can continue.

Various papers have shown that choosing a parallel simulation protocol is a difficult task because it depends on a set of factors concerning the model and the computational platform (Alonso, Frutos and Palacio 1994, Choi and Chung 1995, Xu and Chung 2004). Computational granularity, model partitioning, load balancing, and lookahead are a few factors that can influence the simulation performance.

Attempting to predict the performance of a parallel simulation is a difficult task because different models have different characteristics. Some models can have good speedup, even though some logical processes of the simulation are slower than others.

This paper describes experiments that were carried out with sequential (SMPLX) and conservative distributed simulations (ParSMPLX) to find out the factors that can affect the performance of a simulation. The results show that each logical process of a simulation model has particular characteristics that make it more suitable for a specific protocol (conservative or optimistic) depending on factors such as blocked time, number of null messages exchanged, and the type of blocking that the logical process can experience (necessary or unnecessary).

We explain the factors that affect the simulation performance and use them to demonstrate how the performance can be analyzed in each one of the logical processes. When these metrics are collected and analysed at run-time, an opportunity is created for adapting the simulation for increased speed. The performance we obtained with our method differs from what has been reported in the literature.

The remainder of this paper is organized as follows. Section 2 presents work related to the performance evaluation of parallel simulations. Section 3 describes the functional extensions we used to monitor simulations. Section 4 describes the models and the results we obtained. Finally, in Section 5, we present concluding remarks.

2 RELATED WORK

Various factors must be analyzed to reveal if a parallel simulation has "good speedup" or not. Small simulation models with fine granularity may exhibit better performance using sequential simulation. The use of parallel simulation tend to produce better results in more complex models with high computational granularity (Teo and Tay 1999, Xu and Chung 2004).

Some of the factors that can affect the performance of distributed simulations are: model size, load balancing, computational granularity, communication overhead and model partitioning. In (Xu and Chung 2004), a model is proposed to predict the performance of synchronous discrete-event simulations. This model is used to predict the maximum achievable speedup for given applications and platform characteristics.

Teo and Tay (1999), present a scalable parallel simulation framework called SPaDES (Structured Parallel Discrete-Event Simulation). The paper shows that SPaDES can achieve good speedup for applications with scalable, large models. Teo, Wang and Tay (1999) present a framework for studying the complex performance interactions in parallel simulations considering three main components: simulation model, parallel simulation strategy/protocol and execution platform. The paper analyses how the causality/dependency of events affects the performance of the simulation and determines the potential event parallelism in the simulation models.

Bagrodia et al. (1999) describe the use of COMPASS, a parallel simulator for predicting the performance of programs. Real-world applications and synthetic benchmarks are used to study application scalability, sensitivity to communication latency, and the interplay between factors such as communication patterns and parallel file system caching on application performance.

Teo, Ng and Onggo (2002) evaluate the performance of a conservative simulation using distributed-shared memory for inter-processor communication. They conclude that the performance of parallel simulation is highly dependent on two main factors: the event synchronization overhead and the cost of inter-process communication.

Lee et al. (2005) performed an independent benchmark of the parallel distributed network simulator. The effect of cross-traffic on wall-clock time needed to complete a simulation for a set of basic network topologies was measured. The results were compared with the wall-clock time needed on a single processor. They developed a performance model that can be used as a guideline for designing future simulations.

Teo, Turner and Juhasz (2005) describe a performance analyzer tool developed to predict the execution performance of parallel discrete event simulation programs, using either optimistic or conservative protocols. This analyzer predicts the parallel execution performance based on a sequential simulation run without the development of the parallel implementation.

While the previous articles evaluated the entire simulation, our research considers the evaluation of each logical process isolated from each other. This kind of evaluation can help us to find some weak points of the conservative simulation and allows for some considerations about having two or more protocols simultaneously in a simulation.

3 SIMULATION MONITORS

In order to monitor the parallel simulation and analyze its performance, a modified functional extension based on SMPL (MacDougall 1987) was used. SMPL is an event oriented simulation library used with C language for IBM/PC compatible platform. Using the SMPL source code, a new version was developed for the Unix platform, the SMPLX (Ulson et al. 1999), using dynamical data structures. This approach is for sequential simulations.

Ulson et al. (1999) also made some modifications in SMPLX to adapt it to the conservative protocol CMB with null messages. This approach was developed in IBM/RS6000 and IBM SP2 platforms, using the PVM and PVMe message passing environments. This version was called ParSMPL. Subsequently, Tatsumi (2003) adapted SMPLX and ParSMPL to the Linux operational system. The lookahead of the ParSMPL version was improved by implementing of dynamical lookahead.

These environments allowed for the use of software monitors to obtain important information to evaluate the simulation. One factor is the type of blocking that occurs. Considering conservative parallel simulation with null messages, two types of blocking can occur when a complete message arrives:

- **Necessary:** When the received message is the event to be executed and the event queue is not empty.
- **Unnecessary:** When the received message is not the event to be executed, but one that enables the execution of another event in the event queue.

The unnecessary blocking by null message occurs when a null message arrives and it allows for an event of the event queue to be executed. There are more than these 3 types of blocking, however they do not affect our evaluation in this paper.

These metrics can help to characterize the behavior of each logical process. For example, if many messages are blocked necessarily, it can be concluded that the execution of an optimistic simulation will probably experience many rollbacks. However, this metric is not enough to evaluate the simulation. In addition, the blocked time of each case analyzed was estimated. This time corresponds to the interval between the last executed event and the next one.

This value is important because if a logical process takes more time being blocked necessarily than blocked unnecessarily, then the conservative protocol is probably performing well. If the opposite occurs, an exchange of protocol should be considered.

The blocked time was estimated measuring the interval between the execution of the last event and the reception of a new message to be evaluated. It was considered that the logical processes execute events in the rest of the time.

Another metric is a value that expresses the ratio of the executed time and blocked time. The executed time is the time consumed for running the events and the blocked time is the time consumed for waiting for messages from other logical processes. With these values, it can be observed if the logical process is using more time processing or being blocked. Values higher than 1 show good performances. When this value is smaller than 1, it is important to analyze if there were events in the queue that were ready to be executed (events blocked unnecessarily).

The speedup of the simulation was used to analyze the performance of the simulation comparing the sequential simulation with the distributed simulation (sequential simulation time divided by distributed simulation time).

In our approach, only two logical processes were used because the models were quite simple, however it is important to show many characteristics of conservative distributed simulations.

4 MODELS AND RESULTS

Some models were analyzed to evaluate the performance of each logical process of simulations. The models are a central server (Figure 1), a simplified computational system (Figure 2) and a queue model with two different partitionings (Figure 3 and 4). The central server was decomposed into two logical processes (one with CPU and the other with four disks). The simplified computational system was decomposed into two logical processes (one with CPU and the other with one disk). The queue model was decomposed into two logical processes where the second partition minimizes the communication and has better load balancing.

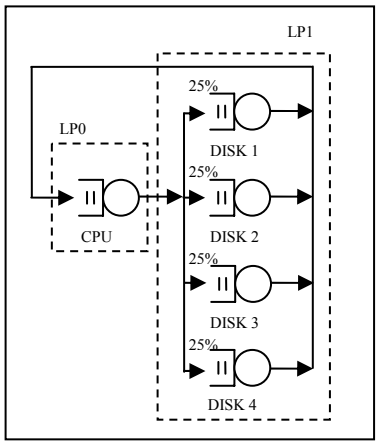

Fig. 1: A Central Server with 2 Logical Processes

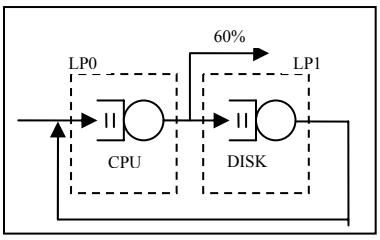

Fig. 2: A Simplified Computational System

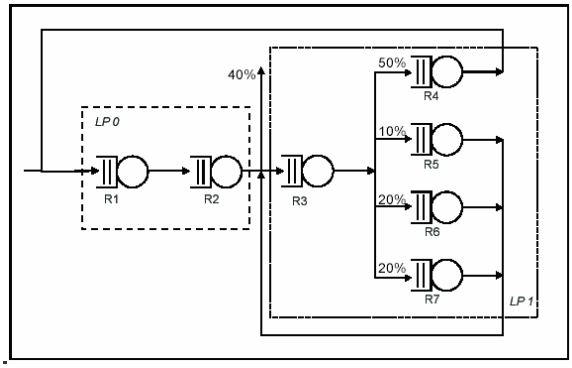

Fig. 3: A Queue Model with 7 Resources

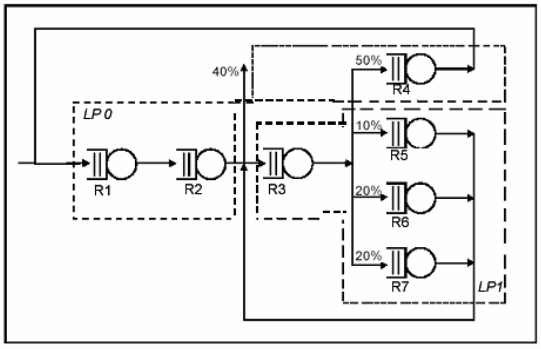

Fig. 4: Same Queue Model with Different Partitioning

The models were executed in a network of computers isolated from external interference. The results of the first model can be observed in Tables 1 and 2, the second model in Tables 3 and 4, the third model in Tables 5 and 6 and the last model in Table 7 and 8. To simulate the computational granularity in each service center, a matrix multiplication task was used. The granularity is represented by the size of the square matrix to be multiplied: a fine granularity (zero and 25), a medium granularity (50 and 75) and a coarse grain (100).

Table 1: Results of Logical Process 0 (Model 1 – Central Server)

Granularity	Speedup	Executing / blocked (LP0)	% blocked time (LP0)	% necessary blocked time	% unneces. blocked time	% neces. blocked complete msg (LP0)	% unneces. blocked compl msg (LP0)	%unneces. blocked null msg (LP0)
0	0.037	0.29	82.82	60.52	17.72	73.79	16.81	80.53
25	0.625	0.76	56.85	31.32	23.66	75.66	6.77	77.05
50	1.037	1.62	45.28	17.06	27.63	75.67	6.77	77.03
75	1.132	2.06	42.55	13.89	28.57	75.66	6.78	77.03
100	1.161	2.23	41.86	12.97	28.89	75.67	6.77	77.03

Table 2: Results of Logical Process 1 (Model 1 – Central Server)

Granularity	Executing / blocked (LP1)	% blocked time (LP1)	% necessary blocked time	% unneces-sary blocked time	% necessary blocked complete msg (LP1)	% unneces. blocked compl msg (LP1)	% unneces. blocked null msg (LP1)
0	0.14	87.96	34.38	52.10	51.36	0.00	98.50
25	0.51	66.37	17.67	47.80	51.03	0.00	95.28
50	0.78	56.29	8.86	47.20	51.05	0.00	95.36
75	0.85	54.04	6.87	47.13	51.05	0.00	95.36
100	0.87	53.34	6.27	47.06	51.05	0.00	95.36

Table 3: Results of Logical Process 0 (Model 2 – Simplified Computational System)

Granularity	Speedup	Executing / blocked (LP0)	% blocked time (LP0)	% necessary blocked time	% unneces. blocked time	% neces. blocked complete msg (LP0)	% unneces. blocked compl msg (LP0)	%unneces. blocked null msg (LP0)
0	0.552	0.654	60.44	18.22	38.88	71.56	0.00	97.74
25	0.819	1.444	40.91	10.32	28.41	71.19	0.00	98.46
50	1.031	3.399	22.73	3.54	18.47	71.46	0.00	98.54
75	1.110	4.693	17.57	1.51	15.79	71.46	0.00	98.54
100	1.129	5.346	15.76	0.83	14.80	71.52	0.00	98.56

Table 4: Results of Logical Process 1 (Model 2 – Simplified Computational System)

Granularity	Executing / blocked (LP1)	% blocked time (LP1)	% necessary blocked time	% unneces-sary blocked time	% necessary blocked complete msg (LP1)	% unneces. blocked compl msg (LP1)	% unneces. blocked null msg (LP1)
0	0.045	95.72	2.51	53.07	9.90	0.00	43.45
25	0.151	86.86	1.71	53.52	9.80	0.00	43.73
50	0.277	78.33	1.12	54.26	9.84	0.00	44.17
75	0.323	75.58	0.95	54.38	9.84	0.00	44.17
100	0.337	74.79	0.89	54.51	9.78	0.00	44.23

Table 5: Results of Logical Process 0 (Model 3 – A Queue Model)

Granularity	Speedup	Executing / blocked (LP0)	% blocked time (LP0)	% necessary blocked time	% unneces. blocked time	% neces. blocked complete msg (LP0)	% unneces. blocked compl msg (LP0)	%unneces. blocked null msg (LP0)
0	0.306	0.32	75.49	15.52	50.30	85.76	0.00	93.00
25	0.809	1.52	39.72	4.92	31.80	85.05	0.00	95.58
50	1.000	2.62	27.59	1.22	25.68	85.05	0.00	95.58
75	1.036	2.96	25.27	0.52	24.51	85.05	0.00	95.58
100	1.046	3.07	24.58	0.32	24.15	85.05	0.00	95.58

Table 6: Results of Logical Process 1 (Model 3 – A Queue Model)

Granularity	Executing / blocked (LP1)	% blocked time (LP1)	% necessary blocked time	% unnecessary blocked time	% necessary blocked complete msg (LP1)	% unneces blocked compl msg (LP1)	% unneces blocked null msg (LP1)
0	0.12	89.03	15.48	71.74	41.64	0.00	99.96
25	0.25	80.17	9.47	70.02	42.04	0.00	99.87
50	0.30	76.84	7.03	69.65	42.04	0.00	99.87
75	0.31	76.18	6.57	69.56	42.04	0.00	99.87
100	0.32	76.01	6.44	69.55	42.04	0.00	99.88

Table 7: Results of Logical Process 0 (Model 3 – A Queue Model with a Different Partitioning)

Granularity	Speedup	Executing / blocked (LP0)	% blocked time (LP0)	% necessary blocked time	% unneces. blocked time	% neces. blocked complete msg (LP0)	% unneces. blocked compl msg (LP0)	%unneces. blocked null msg (LP0)
0	0.335	0.43	170.78	420.22	602.03	92.34	0.00	91.56
25	0.854	2.19	50.96	230.43	734.03	92.23	0.00	95.85
50	1.041	4.33	11.32	169.21	901.41	92.23	0.00	95.85
75	1.076	5.09	3.90	157.96	962.18	92.23	0.00	95.85
100	1.087	5.35	1.78	154.68	982.84	92.23	0.00	95.85

Table 8: Results of Logical Process 1 (Model 3 – A Queue Model with a Different Partitioning)

Granularity	Executing / blocked (LP1)	% blocked time (LP1)	% necessary blocked time	% unnecessary blocked time	% necessary blocked complete msg (LP1)	% unneces. blocked compl msg (LP1)	% unneces. blocked null msg (LP1)
0	0.16	86.40	17.64	67.16	43.78	0.00	99.92
25	0.21	82.44	12.91	68.94	44.07	0.00	99.92
50	0.23	81.35	11.02	70.19	44.07	0.00	99.92
75	0.23	81.08	10.66	70.37	44.07	0.00	99.92
100	0.23	80.98	10.56	70.40	44.07	0.00	99.92

In logical process 0 of model 1, it can be observed that the ratio of executed time to blocked time is higher than 1 for values of granularity higher than 50. The blocked time decreases as granularity increases. Although there is a high percentage of unnecessary blocked null messages, a large amount of complete messages that were blocked necessarily were found, indicating that the conservative approach is more appropriate. If an optimist approach were used, a high quantity of messages could be executed incorrectly, resulting in more rollbacks.

On the other hand, in logical process 1 of the same model, a large percentage of unnecessary blocked null messages and almost 50% of unnecessary blocked time can be observed. The percentage of complete messages blocked necessarily is not as high (51%). This shows that the change of protocol should be considered because much time is being wasted waiting for messages that just update the clock channel.

Model 2 shows a similar behavior to model 1. In logical process 0, there is a high percentage of complete messages

(more than 70%) in all granularities and for values of granularity higher than 25 there is a ratio of executed time to blocked time higher than one, showing that this logical process performs well.

Logical process 1 has a different behavior because all granularities have a ratio of executed time to blocked time lower than one, resulting in a bad performance. However, in this case, there is a low percentage of complete messages that caused necessary blocking (10%) and a high percentage of null messages that caused unnecessary blocking (44%), showing that a change of protocol to the optimistic one can improve the simulation performance.

Figures 3 and 4 show the same model with different partitioning. In logical process 0 of both models, the blocked time decreases as granularity increases. It is different from logical process 1, which remains with a high percentage of blocked time in both models.

Checking the unnecessary blocked time in logical process 0, it can be observed that it decreases as granularity increases. This behavior is different from logical process 1 where the percentage is almost the same.

The outcomes show that logical process 0 can perform well when granularity is increased, while logical process 1 remains blocked most of the time despite the increase of granularity.

By analyzing the ratio of executed time to blocked time, it can be concluded that the performance of logical process 0 increases as granularity increases and a large amount of complete messages is blocked necessarily (more than 80%). On the other hand, analyzing the logical process 1, it can be observed that the percentage of complete messages blocked necessarily was low (40%) and the percentage of null messages blocked unnecessarily was high (99%). These results, in addition to the high percentage of unnecessary blocked time (70%), show that the execution of the messages in the queue could have resulted in a better performance.

Although the speedup of model 4 is better than model 3, it can be considered poor in both cases, showing that this kind of model does not perform well with the conservative simulation. The use of sequential simulation or another approach of distributed simulation could perform better.

From all the results obtained, it can be observed that:

- Blocked time and unnecessary blocked time decreases as the computational granularity increases.
- The ratio of executed time to blocked time increases as the granularity increases.
- The percentage of complete blocked messages that was necessary in relation to all complete messages that arrived in the logical process can be related to the blocked time. If it has a high percentage of these types of messages (more than 50%) and not much blocked time (a ratio of executed time to blocked time higher than one) then the conservative approach is performing well
- In the same way, if most of the null messages were unnecessarily blocked (a percentage of null messages that caused unnecessary blocking higher than 50%), a low percentage of complete messages causes necessary blocking (fewer than 50%) and the ratio of executed time to blocked time was lower than 1, then the change of the protocol to optimistic can be considered.
- In some cases, if the ratio of executed time to blocked time is lower than 1 and a high percentage of complete messages causes necessary blocking, then the performance is not good. On the other hand, changing to the optimistic protocol is also not appropriate. In this case, the use of sequential simulation is more appropriate.

From the examples used to test, it can be observed that the number of necessary and unnecessary blocking events does not change with granularity, therefore this metric cannot be used alone to evaluate a simulation. However, this information can be used with blocked time, resulting in a good parameter to evaluate a simulation.

The parallel simulation with fine granularity did not perform well in the experiments reported in this paper. In this case, the sequential simulation is more appropriate. This occurs because logical processes spend more time being blocked than executing the simulation.

5 CONCLUSIONS

This paper shows that the performance of each logical process of the simulation can be analyzed and evaluated and that in the same simulation different logical processes can show different performances.

The analysis was possible by adding software monitors to the simulation code. The monitors collected some values that were used to estimate each logical process performance in execution time. The values obtained were used to define some metrics that show the performance of the logical processes.

These metrics demonstrate that logical processes have distinct characteristics that make them more suitable for a specific protocol. In the analyzed cases, one logical process performs well using the conservative approach, however the other could be more efficient using the optimist approach.

On the other hand, the adoption of an optimistic protocol to the whole simulation is not a good solution because the occurrence of rollbacks could degrade the simulation performance. Taking this into account, our results lead us to conclude that each one of the logical processes has a different behavior which makes it more suitable for a differ-

ent protocol. If the same simulation with different protocols can be executed, a better performance can be obtained.

In further work, we plan to evaluate complex models with ParSMPLX and investigate the performance of the optimistic protocol.

REFERENCES

Alonso, J. M., A. A. Frutos, and R. B. Palacio. 1994. Conservative and optimistic distributed simulation in massively parallel computers: a comparative study. *Proceedings of the 1ST International Conference On Massively Parallel Computing Systems*, 528-532.

Bagrodia, R., E. Deeljman, S. Docy, and T. Phan. 1999. Performance prediction of large parallel applications using parallel simulations. *Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (PPoP '99).

Bauer, D., G. Yaun, C. D. Carothers, M. Yuksel, and S. Kalyanaraman. 2005. Seven-O'Clock: a new distributed GVT algorithm using network atomic operations. *Proceedings of the 18th Workshop on Parallel and Distributed Simulation* (PADS' 2005).

Bononi, L., M. Bracuto, G. D'Angelo, and L. Donatiello. 2005. Concurrent replication of parallel and distributed simulations. *Proceedings of the 18th Workshop on Parallel and Distributed Simulation* (PADS' 2005).

Bruschi, S. M., R. H. C. Santana, M. J. Santana, and T. S. Aiza. 2004. An automatic distributed simulation environment. *Proceedings of the 2004 Winter Simulation Conference*. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Choi, E., and M. J. Chung. 1995. An important factor for optimistic protocol on distributed systems: granularity. *Proceedings of the 1995 Winter Simulation Conference*. 642-649. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Curry, R., C. Kiddle, R. Simmonds, and B. Unger. 2005. Sequential performance of asynchronous conservative PDES algorithms. *Proceedings of the 18th Workshop on Parallel and Distributed Simulation* (PADS' 2005).

Dongarra, J., I. Foster, G. C. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. 2002. *Sourcebook of parallel computing*. Morgan Kaufmann Publishing.

Fujimoto, R. M. 2003. Distributed simulation systems. *Proceedings of the 2003 Winter Simulation Conference*. 124-134. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Kumar, V., A. Grama, A. Gupta, and G. Karpis. 2004. *Introduction to parallel computing design and analysis of parallel algorithms*. Benjamin-Cummings Publishing.

Lee, J. S., T. Luu, and V. K. Konangi. 2005. Design of a satellite cluster system in distributed simulation. *Simulation* 81: 57-66.

Lee, S.; J. Leaney, T. O'Neill, and M. Hunter. 2005. Performance benchmark of a parallel and distributed network simulator. *Proceedings of the 18th Workshop on Parallel and Distributed Simulation* (PADS' 2005).

MacDougall, M. H. 1987. *Simulating computing systems - techniques and tools*. The MIT Press.

Tatsumi, E. S. 2003. Avaliação e aprimoramento de uma implementação para simulação distribuída conservativa visando utilização em um ambiente automático. MSc. Dissertation, Department of Computer and Statistic, University of São Paulo, Brazil.

Teo, P., S. J. Turner, and Z. Juhasz. 2005. Optimistic protocol analysis in a performance analyser and prediction tool. *Proceedings of the 18th Workshop on Parallel and Distributed Simulation* (PADS' 2005).

Teo, Y. M., Y. K. Ng, and B. S. S. Onggo. 2002. Conservative simulation using distributed-shared memory. *Proceedings of the 16th Workshop on Parallel and Distributed Simulation* (PADS' 2002).

Teo, Y. M., and S. C. Tay. 1999. Performance evaluation of a parallel simulation environment. *Proceedings of the 32nd Annual Simulation Symposium*.

Teo, Y. M., H. Wang, and S. C. Tay. 1999. A Framework for analyzing parallel simulation performance. *Proceedings of the 32nd Annual Simulation Symposium*.

Ulson, R. S., J. C. M. Morselli Jr., R. H. C. Santana, and M. J. Santana. 1999. Conservative distributed simulation on portability platforms: the CMB protocol behavior. *Proceedings of the International Conference Applied Modeling and Simulation* (IASTED).

Xu, J., and M. J. Chung. 2004. Predicting the performance of synchronous discrete event simulation. *IEEE Transactions on Parallel and Distributed Systems*, 15, 12: 1130-1137.

AUTHOR BIOGRAPHIES

CÉLIA L. O. KAWABATA is a lecturer in the Centro Universitário Central Paulista (UNICEP). She has a B.Sc., M.Sc. and Ph.D. in Computer Science from USP, in 1997, 2000 and 2005, respectively. Her research interests include distributed simulation and performance evaluation. She is a member of the Brazilian Computer Society (SBC). Her e-mail address is <celiak@gmail.com>.

REGINA H. C. SANTANA is an Associate Professor in the Computer Science Department at the University of São Paulo (USP) and co-ordinator of the B.Sc Computer Science course. She has a B.Sc. in Electrical Engineering from USP (1980), an M.Sc. in Computer Science from USP (1985) and a Ph.D. in Electronics and Computer Science from the University of Southampton, UK (1990). Her research interests include distributed simulation, modeling techniques and performance evaluation. She is a member

of the Brazilian Computer Society (SBC). Her e-mail address is <rcs@icmc.usp.br>.

MARCOS J. SANTANA is an Associate Professor in the Computer Science Department at the University of São Paulo (USP). He has a B.Sc. in Electrical Engineering from USP (1980), an M.Sc. in Computer Science from USP (1985) and a Ph.D. in Electronics and Computer Science from the University of Southampton, UK (1990). His research interests include modelling techniques, performance evaluation and parallel/distributed computing. He is a member of the SBC. His e-mail address is <mjs@icmc.usp.br>.

SARITA M. BRUSCHI is a lecturer in the Computer Science Department at the University of São Paulo (USP). She has a B.Sc. in Computer Science from UNESP (1994), and an M.Sc. and Ph.D. in Computer Science from USP in 1997 and 2002, respectively. Since 2003 she has been working at the Distributed System and Concurrrent Programming Laboratory in the Institute of Mathematical and Computational Sciences. Her special fields of interest include distributed simulation, performance evaluation and parallel/distributed computing. She is a member of the Brazilian Computer Society (SBC). Her e-mail address is <sarita@icmc.usp.br>.

KALINKA R. L. J. CASTELO BRANCO is a lecturer in the Fundação de Ensino "Eurípides Soares da Rocha" (UNIVEM). She has an M.Sc. and Ph.D. in Computer Science from USP in 1999 and 2004, respectively. Her research interests include distributed systems, performance evaluation, load balance and computer networks. She is a member of the Brazilian Computer Society (SBC). Her e-mail address is <kalinka@fundanet.br>.