A Configuration Control Mechanism Based on Concurrency Level for a Reconfigurable Consistency Algorithm

Christiane V. Pousa[1]
[1]Computational and Digital Systems Group - Pontifical Catholic University of Minas Gerais - Belo Horizonte – Brazil
pousa@ieee.org

Luís F. W. Góes[2]
[2]Computational and Digital Systems Group - Pontifical Catholic University of Minas Gerais - Belo Horizonte – Brazil
lfwgoes@yahoo.com.br

Carlos A. P. S. Martins[3]
[3]Computational and Digital Systems Group - Pontifical Catholic University of Minas Gerais - Belo – Brazil
capsm@ieee.org

Abstract

A Reconfigurable Consistency Algorithm (RCA) is an algorithm that guarantees the consistency in Distributed Shared Memory (DSM) Systems. In a RCA, there is a Configuration Control Layer (CCL) that is responsible for selecting the most suitable RCA configuration (behavior) for a specific workload and DSM system. In previous works, we defined an upper bound performance for RCA based on an ideal CCL, which knows apriori the best configuration for each situation. This ideal CCL is based on a set of workloads characteristics that, in most situations, are difficult to extract from the applications (percentage of shared write and read operations and sharing patterns). In this paper we propose, develop and present a heuristical configuration control mechanism for the CCL implementation. This mechanism is based on an easily obtained applications parameter, the concurrency level. Our results show that this configuration control mechanism improves the RCA performance in 15%, on average, compared to other traditional consistency algorithms. Furthermore, the CCL with this mechanism is independent from the workload and DSM system specific characteristics, like sharing patterns and percentage of writes and reads.

I. Introduction

In a distributed system, when an application has parallel access on shared objects, the users may use a model that helps the management of these accesses and simplifies the programming. Distributed shared memory is one of these models that have received considerable attention in the last years [1] [2]. DSM is an abstraction that provides an illusion of a shared memory in a parallel and distributed system. In object based DSMs, the operations semantics (Consistency) guarantee that objects will be consistent for the process during the workload execution [1] [2].

A consistency algorithm can be defined as a contract between the application and the shared objects. This contract has the rules about how and when a process of an application can access the shared objects [1] [2].

The Reconfigurable Consistency Algorithm (RCA) was proposed to improve flexibility and performance of the object-based DSM systems. RCA is an algorithm that guarantees the consistency in an object-based DSM Systems. It can have its behavior modified considering the workloads and DSM Systems characteristics. So, it can adapt to the workloads and DSM Systems characteristics, becoming more flexible and increasing performance [1] [2] [3].

In a RCA, there is a Configuration Control Layer (CCL) that is responsible for selecting the most suitable RCA configuration (behavior) for a specific workload and DSM system. Thus, the functioning of the RCA and the configurations are decided within this layer. Its decisions are made upon input parameters, dynamic workload information, commands from the operating system, user's choice etc. The CCL design is a complex key problem in the design of a reconfigurable algorithm [1] [2] [3] [4] [5] [6].

Until now, we defined a theoretic upper bound performance for RCA based on an ideal CCL, which knows apriori the best configuration for each situation. This CCL is based on a set of workloads characteristics that are difficult to extract from the applications [1] [2] [3].

In this paper we propose, develop and present a heuristical configuration control mechanism for the CCL implementation. This mechanism is based on the concurrency level of the DSM workloads. The concurrency level represents the amount of

1-4244-0910-1/07/$20.00 c2007 IEEE

concurrency that the application will have during its execution in the DSM system. It can be easily obtained from the number of shared objects and tasks of the applications. So, with this mechanism, the CCL can be implemented without apriori knowledge about all possible situations (ideal condition), which is infeasible in a real system.

The main objectives of this paper are: to propose and present a configuration control mechanism, to implement and analyze the performance of it in a reconfigurable consistency algorithm using simulation. The main goals are: the development of the configuration control mechanism in a simulation tool and its use on a RCA.

This paper is organized as follow: in section 2, we present some important related works; in section 3 we present some concepts about the reconfigurable consistency algorithm RCA; in section 4, we present our configuration control mechanism; sections 5 and 6 describes and present the experimental method and results and in section 7, we present our conclusions and describes some future works.

II. Related Work

In the last years, many consistency algorithms have been proposed [1] [2] [7] [8] [9] [10] [11] [12], but in this paper, we will discuss only some papers that are more relevant and close to our work [1] [2] [7] [8]. In papers [1] [2], we propose and present a reconfigurable object consistency. In [7] [8], adaptive consistency algorithms are presented. And, in [10] [11] traditional consistency algorithms are presented.

In the works [1] [2], we propose and present a reconfigurable object consistency that was represented by a reconfigurable algorithm. In this reconfigurable consistency, the CCL is based on the percentage of writes instructions and on the number of objects of the workload. The objects number is easy to be obtained from the workloads, but the percentage of writes instructions is difficult. To obtain the percentage of write instructions in a workload, it has to be analyzed by a program that identifies this percentage. Furthermore, the number of objects and percentage of write instructions change from application to application. So, this is expensive and difficult to obtain.

In [7], a flexible consistency algorithm is proposed and implemented. It uses a different algorithm depending on the selected parameter. The consistency algorithm implements three-consistency models, but it just uses the traditional implementation of each one. Moreover, just one parameter (the consistency model) is used to change the consistency. So, the consistency used has to be chosen by the users and not based on the workloads and DSM Systems characteristics like in [1] [2].

In [8], a hybrid software DSM protocol is presented. This DSM can adapt to sharing patterns. The adaptation is made considering just the access policy and the synchronization primitives. Furthermore, their consistency model can only be adapted for some release consistency model variations. The adaptation is done based on the access patters that are difficult to be obtained from the workloads in most of the cases.

In [10], a traditional sequential consistency algorithm is proposed and presented. In [11], an atomic consistency algorithm is proposed and its implementation is presented. In both works, some qualitative and quantitative results are presented.

In the works [1] [2], they present a reconfigurable object consistency implemented with an ideal CCL that knows apriori the best configuration for each situation. In papers [7] [8] [10] [11], the authors present adaptable and fixed consistency algorithms. So, regarding to consistency details, the works that are closest to our work are [7] [8] [9] [10] [11] [12] and regarding to CCL the closest works to our approach are [1] [2].

III. Reconfigurable Consistency Algorithm

RCA is a reconfigurable consistency algorithm for asynchronous architectures that execute an object-based software DSM [1] [2]. It manages the state of a set of shared objects. RCA can have its behavior reconfigured considering the workloads and DSM systems characteristics. So, it can adapt to the workload and architecture characteristics, becoming more flexible and increasing performance.

RCA is composed of three layers: Basic Layer (BL), Reconfigurable Layer (RL) and Configuration Control Layer (CCL). The BL consists of a frame set and data structures. A frame represents a part or phase of an algorithm. A data structure may be a list, a queue, an array or some structure that stores data. For example, a wait queue (data structure) stores operations (data). The Reconfigurable Layer represents a configuration or an instance of the BL, in which every frame is filled out with one compatible constructive block at a certain moment. A constructive block is a possible implementation that can fill out with a specific frame. The Configuration Control Layer chooses the constructive blocks that will fill out each frame at a given moment, thus it controls the configuration swapping. The choice is made based on entry parameters [2].

RCA has five parts (an event ordering policy frame, a constraint policy frame, a coherence protocol frame, a replication protocol frame and an access policy frame) and some implementations (blocks) that are combined to reconfigurable it. So, it can have its behavior assuming any type of consistency. The actual version of RCA can be reconfigurable to assume atomic or sequential consistency algorithms variations [1] [2].

IV. Configuration Control Mechanism

As we said before, RCA provides flexibility and performance improvement for the consistency algorithms. An

important RCA layer is CCL, because it is responsible for the RCA reconfiguration process, and for the flexibility and performance improvements.

The Configuration Control Layer (CCL) is responsible for selecting and swapping the constructive blocks that fill in the RCA frames at a given moment. So, it is responsible for generating the Reconfigurable Layer (instance) from the Basic Layer. The decisions about the selection and swap might be made based on: input parameters, dynamic workload information, commands from the operating system, user's choice etc.

The input parameters used in the previous version [1] [2] of CCL were a set of metrics (response time, number of messages and communication time) that represents the DSM system characteristics and a set of some DSM workload characteristics (number of objects, percentage of write instructions, number of tasks and sharing patterns). Most of these DSM system and workload characteristics are difficult to obtain in practice. Because, these characteristics are related to different computer system levels, like the programming model, language, algorithm, architecture, etc. However, there are some DSM system and workload characteristics that can be extract from the applications through the use of profiling tools or by the user specification [13]. Among these characteristics we can cite the number of tasks and number of shared objects.

Based on the number of tasks and number of shared objects, we propose a new configuration control mechanism for the CCL that we called concurrency level heuristic. The concurrency level heuristic is based on the amount of shared objects for the set of tasks of an application. It represents the amount of concurrency that the application will have during its execution in the DSM system. So, the concurrency level can influence the performance and the parallelism level of the applications. Furthermore, a same concurrency level can represent a set of applications, because different applications can have the same concurrency level.

To create the concurrency level heuristic, we defined the application concurrency level (cl) parameter. The cl of an application is defined as the number of objects divided for the number of tasks (Equation 1).

$$ConcurencyLevel = \frac{NumberofObjects}{NumberofTasks} \quad \text{(Equation 1)}$$

The concurrency level of an application can be minimum, medium and maximum. In minimum *cl* applications, the number of tasks is always the same that the number of objects. In medium *cl* applications, we have at least one object for each two tasks. And, in the maximum *cl,* the number of objects for all applications at the same time is one. So, for a minimum *cl* any consistency algorithms are good enough. On the other hand, for medium and maximum *cl* weaker consistency algorithms are better than the strong ones.

The heuristic uses the cl parameter of each application in the $getConfigurationForcl_k$ () function to define the best configuration for that application. With the cl, the new CCL became more generic and smaller than the previous CCL, because different types of application can have the same cl. In figure 1 we present the new CCL that use cl (just one value) instead of W (a set of values), as the CCL presented in [1] [2].

```
function configure
    Input cl Output Cp
    Switch (cl)
      Case cl0
        Cp ← getConfigurationForcl0()
      End
      .
      .
      .
      Case clk
        Cp ← getConfigurationForclk()
      end
    end
end
```

Fig. 1. The configuration control layer with the Configuration Control Mechanism

V. EXPERIMENTAL METHOD

In this section, we describe the metrics, DSM system architecture and workload used for the configuration control mechanism performance evaluation. Afterwards, we describe the experimental design in which we highlight consistency algorithms that will be used in the performance analysis of the configuration control mechanism.

In order to analyze the configuration control mechanism, we can use different metrics. The most common is the response time metric [1] [2] [3] [4] [5]. The mean job response time, defined in Eq.2, is the mean time interval between the submission and end of a job.

$$MeanRespTime = \frac{\sum JobEndTime - JobSubmissionTime}{NumberofJobs}$$

(Equation 2)

The DSM system architecture is a cluster composed of 8 nodes interconnected by a Fast Ethernet switch. Each node has an object software DSM. In Table 1, we see the main values of the cluster's characteristics, obtained from benchmarks and performance libraries (Sandra 2003, PAPI 2.3 etc.). We modeled our environment in ClusterSim, a simulation tool [14] [15]. In [14], the DSM and the traditional (proposed) consistency models (TCM) implementations were verified, in ClusterSim.

Table 1. Cluster characteristics and respective values

Characteristic	Value	Characteristic	Value
Number of nodes	8	Network	Fast Ethernet
Processor Frequency	0.938 GHz	Network Latency	0.000179 s
Cycles per Instructions	0.9997105	Max. Segment Size	1460

Primary Memory Transfer Rate	1114.6 MB/s	Network Bandwidth	11.0516 MB/s
Secondary Memory Transfer Rate	23.0 MB/s	Protocol overhead	58 bytes
Invalidate Message Size	96 bytes	Update Message Size	1K – 4K

DSM applications can be categorized into three broad categories, namely fork-join, run-to-complete and iterative [16]. Each one of these categories has different characteristics. The interactive applications exhibit a regular program behavior and for this reason we will simulate different types of them.

In order to simulate different applications, we choose some values and characteristics (number of shared objects, number of tasks, access patters and number of writes and reads) that are important in an object-based software DSM. So, we create our synthetic workload. Our workload is a set of six sub-workloads. Each one of these sub-workloads is composed of ten applications. The applications have shared objects to represent some common structures (Matrix or Vector), used in some common parallel problems, like Matrix Multiplication, Image Convolution, Quick Short etc. To cover the maximum number of workloads, we used different access patterns, different number of objects and tasks and different number of interactions and instructions.

After the choice of the characteristics, we modeled the workload applications based on these characteristics and on the concurrency level (*cl*) in ClusterSim. Table 2 presents the sub-workloads number of applications considering the three different concurrency levels. The complete description of the workload can be found in [3].

Table 2. Workload characteristics

Workload	Number of Appl. minimum *cl*	Number of Appl. medium *cl*	Number of Appl. Maximum *cl*
Sub-Workload 1	2	4	4
Sub-Workload 2	4	2	4
Sub-Workload 3	4	4	2
Sub-Workload 4	0	2	8
Sub-Workload 5	8	0	2
Sub-Workload 6	2	8	0

In order to analyze the performance of our configuration control mechanism, we compare the RCA with CCL implemented with the proposed mechanism with four consistency algorithms. These algorithms are: the two algorithms presented in [10] and [11], the reconfigurable consistency algorithm implemented with a theoretic upper bound CCL presented in [1] [2] and an algorithm (the best on average) generate from the configurations of the RCA.

We will call the works presented in [10] and [11] as Algorithm 1 and 2. The RCA implemented in [1] and [2] will be called as RCA_Ideal, because it is a theoretic upper bound for the maximum speedup that RCA can generate. And, the RCA implemented with the proposed configuration control mechanism will be called RCA_*cl*. Finally, the best (on average) algorithm generate from the RCA configurations will be called Algorithm 3.

To compare the algorithms 1, 2, 3 and RCA_ideal with our RCA_*cl,* we model them in ClusterSim and simulate the same workload for the five algorithms. So, considering the workload and the five consistency algorithms (Algorithm 1, 2, 3, RCA_Ideal and RCA_*cl*) we made 300 simulations (5 algorithms x 60 applications simulations).

VI. Simulation Results

In this section we present the response time results obtained from the simulations. For the performance analysis of the proposed mechanism, we compare the RCA implemented with this mechanism with other two traditional consistency algorithms, proposed in [10] and [11]. Furthermore, we show the RCA_*cl* thresholds, comparing it with what we call RCA_Ideal and with the best configuration generate from RCA frames combination. In the end of the section, we present the cumulative response time for the simulated algorithms and workload.

A. Traditional Consistency Algorithms

In figure 2, we present the response time for each sub-workload from the simulated workload for the algorithms 1, 2 and RCA_*cl*. As we can observe in fig. 2, the algorithm 1 presents the worst results. This algorithm is an atomic consistency and is very strong. So, it not allows the tasks to access the shared objects in parallel, which decreases the performance of the applications and DSM systems.

We can also observe in this figure that in sub-workloads 2, 3 and 6 (W2, W3 and W6) the RCA_*cl* and Alg. 2 have the same response time. In the W2 and W3 the applications distributions are very uniform (table 2). So, the sub-workloads became homogeneous, and the consistency algorithm use in their execution does not change their performance. Considering the sub-workload 6 (W6), these two algorithms have the same results because there are not applications with maximum *cl.* W6 is composed of 8 applications with minimum *cl.* The consistency algorithm does not influence the performance of these applications, because the concurrency level is very small.

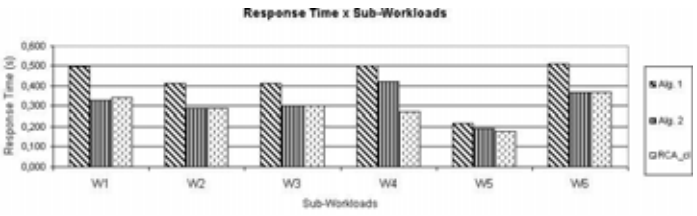

Fig. 2. Response Time for the Algorithms 1, 2 and CCL

For the sub-workload 1 (W1), the Alg.2 has the best results. The RCA_*cl* have a large response time in one of the

applications. In the others applications of W1, RCA_*cl* and Alg. 2 have the same results. So, because of a wrong RCA_*cl* configuration choice for one of the applications the Alg. 2 became better.

With sub-workloads 4 and 5 (W4 and W5) the RCA_*cl* have the best results. In these sub-workloads the presence of the applications with maximum *cl* gives to Alg. 2 the worst results. The applications with maximum cl are responsible for the RCA_*cl* speedup, because there is a lot of concurrency and the RCA_*cl* have to find the best performance for each concurrency level. Furthermore, Alg.2 is not good for applications with maximum *cl* because it do not allow multiple writes or reads, which decrease the system performance.

B. RCA_cl Thresholds

In Fig. 3 we present the response time for RCA_*cl* compared with Alg. 3 and RCA_Ideal. From all algorithms, Alg. 3 present the best response time for the simulated workload. Because of this, we compare its performance to the RCA_Ideal and RCA_*cl*.

The bottom line represents the response time for the RCA_Ideal. As we can see, this algorithm has the best results for all sub-workloads. However, the simulation of RCA_Ideal is important to show that RCA_*cl* have results very closer to it. As we can observe in fig. 3, RCA_*cl* have better results than the best algorithm (Alg. 3). Here, it is also important to note that RCA_*cl* is much better in W4. This sub-workload has eight maximum *cl* applications, what means that our concurrency level heuristic is better for workloads with much concurrency. Considering that this kind of applications (maximum *cl*) is presented in almost all DSM workloads and benchmarks [17] [18], RCA_*cl* can be used to improve the performance of DSM applications.

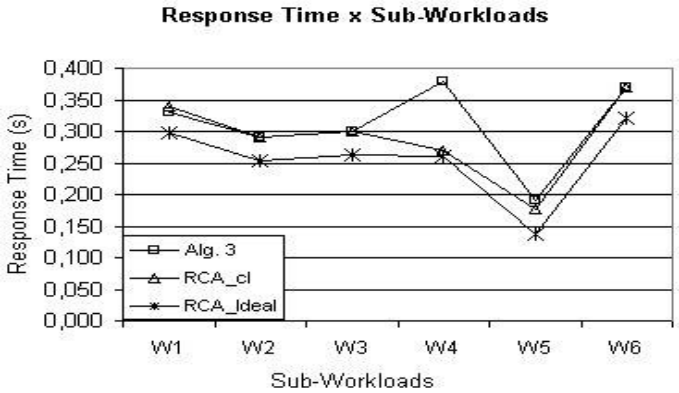

Fig. 3. Response Time for Alg.3, RCA_cl and RCA_Ideal

C. Cumulative Response Time

In fig.4 we present the cumulative response time for the analyzed algorithms (Alg.1, Alg. 2, Alg. 3, RCA_*cl* and RCA_Ideal). The cumulative response time is presented for the sub-workloads in which RCA_*cl* do not have the same results of the others algorithms (W1, W4 and W5).

As we can see in the figure, RCA_*cl* have results very closer to the RCA_Ideal. This means that, considering the compared algorithms RCA_*cl* is the closest to the maximum possible speedup (RCA_Ideal).

Another important thing to observe in this figure is that Alg. 1 and 2 are far way from the RCA_Ideal. They are closer to the Alg. 3 and RCA_*cl*, but they have worse results than Alg. 3 and RCA_*cl*. It is important to say that Alg.3 is one of the possible RCA_*cl* configurations and it wasn't proposed in literature. So, for W1, W4 and W5 the algorithms 1 and 2 do not present a good performance as Alg. 3 and RCA_*cl*.

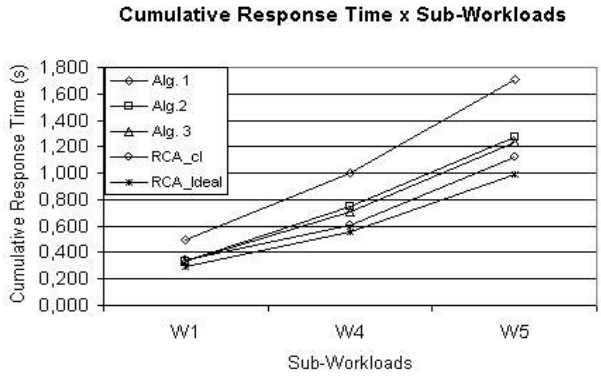

Fig. 4. Cumulative Response Time for Alg.1, Alg.2, Alg.3, RCA_cl and RCA_Ideal

In table 3 we present the response time for each sub-workload and algorithm. We also present the response time for the complete workload execution for each algorithm.

Analyzing this table, we can conclude that considering the best algorithm (algorithm that present the smallest response time – Alg. 3), RCA_*cl* presents a speedup of 6%. And considering all the algorithms, the RCA_*cl* presents a speedup of 15%, on average. The RCA_Ideal present 17% of speedup in relation of the best algorithm (Alg. 3). And, a speedup of 25% considering all algorithms. So, considering that RCA_Ideal is a theoretic upper bound, RCA_*cl* presents a good speedup. All the simulations and results analysis can be found in [3].

VII. Conclusions and Future Work

In this paper, we presented a configuration control mechanism based on concurrency level. Moreover, we analyzed a workload, in which the use of mechanism achieved very good results (more than 15% speedup). One of the most important results, in all simulations, was to show that depending on the workload, the RCA implemented with the proposed configuration control mechanism have results very closer to the RCA_Ideal (Theoretic Upper Bound performance of RCA) and better than the best algorithm generate from RCA

configurations (Alg. 3).

Our mechanism is more independent, because depends just of the concurrency level and not of a set of workload characteristics like the CCL in RCA_Ideal presented in [1] [2]. Thus, RCA_Ideal cannot be generic for all situations, because the percentage of write instructions changes from application to application. Due to the high heterogeneity and stochastic behavior of the workloads, the configuration control mechanism based on concurrency level appears as an alternative solution, providing more independence from specific workload characteristics.

The **main contributions** of this paper are: the presentation, implementation and performance analysis of a configuration control mechanism based on concurrency level, comparing it with other consistency algorithms for different workloads.

As future works and open researches we can highlight: give more independence for the configuration control mechanism; compare RCA implemented with the proposed configuration control mechanism with other consistency algorithms, simulation of different workloads and real tests.

REFERENCES

[1] Pousa C. V., Góes L. F. W., Martins C. A. P. S., Reconfigurable Object Consistency Model, 7th Advances in Parallel and Distributed Computational Models, IPDPS, 2005.

[2] Pousa C. V., Góes L. F. W., Penha D. O., Martins C. A. P. S., Reconfigurable Sequential Consistency Algorithm, in 12th Reconfigurable Architecture Workshop, IPDPS, 2005.

[3] [3] Reconfigurable Object Consistency Model Project – (ROCoM´s Project). URL: http://planeta.terra.com.br/negocios/christianepousa/rocom/index.htm

[4] Ramos L. E. S., Martins C. A. P. S., Reconfigurable Collective Communication MPI Functions. In V Workshop on High Performance Computational Systems, 2004, pp.176-183 . (in Portuguese)

[5] Góes L. F., Proposal and Development of a Reconfigurable Parallel Job Scheduling, M.Sc. Thesis Graduation Program in Electrical Engineering, Pontifical Catholic University of Minas Gerais, 2004. (in Portuguese)

[6] Góes L. F.W. and Martins C.A.P.S., Reconfigurable Gang Scheduling Algorithm, 10th Workshop on Job Scheduling Strategies for Parallel Processing, LNCS, 2004.

[7] Jiménez E., Fernández A., and Cholvi V., A Parametrized Algorithm that Implements Sequential, Causal, and Cache Memory Consistency. Workshop on Parallel, Distributed and Network-based Processing, 2002, pp.437-444.

[8] Monnerat L. R. and Bianchini R., Efficiently Adapting to Sharing Patterns in Software DSMs. Proceedings of the 4th IEEE International Symposium on High-Performance Computer Architecture, 1998.

[9] Lamport L., How to make a multiprocessor computer that correctly executes multiprocess programs. IEEETrans. Comput.1979, pp. 28:690-691.

[10] Zhou J.Z., Mizuno M., and Singh G., A Sequentially Consistent Distributed Shared Memory, In 5th Int'l Conference on Computing and Information, 1993, pp.165-169.

[11] Torres-Rojas F. J., Ahamad M., Raynal M., Timed consistency for shared distributed objects. Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, 1999, pp. 163-172.

[12] Adve S.V. and Hill M.D., Weak Ordering - A New Definition. Proceedings of the 17th Annual International Symposium on Computer Architecture, 1990, 2-14.

[13] Brorsson M. and Kral M, Performance Tuning Software DSM Applications using Visualisation. The Journal of Supercomputing - Kluwer Academic Publishers, 1999, vol. 13, no. 3, pp. 249-265(17).

[14] Pousa C. V., Ramos L. E. S., Goes L. F. W., Martins C. A. P. S., Extending ClusterSim with MP and DSM Modules, In International Symposium on High Performance Computational Science and Engineering, 2004.

[15] Góes L. F. W., Ramos L. E. S., Martins C. A. P. S., ClusterSim: A Java Parallel Discrete Event Simulation Tool for Cluster Computing. IEEE International Conference on Cluster Computing, 2004.

[16] Liu Y., Liang T., Kuo Z. and Shich C., Involving Memory Resource Consideration into Workload Distribution for Software DSM Systems, in DSM Workshop, 2004.

[17] R. Pozo and B. R. Miller, The SciMark 2 Website. Web Site accessed on 18 April 2005 http://math.nist.gov/scimark2/

[18] Woo S. C., Ohara M., Torrie E., Singh J. P., and Gupta A., The SPLASH-2 Programs: Characterization and Methodological Considerations. 22nd Annual International Symposium on Computer Architecture 24-36, 1995.