Reconfigurable Object Consistency Model

Christiane V. Pousa, Luís F. W. Góes, and Carlos A. P. S. Martins
Graduation Program in Electrical Engineering – Computational and Digital Systems Group
Pontifical Catholic University of Minas Gerais - Belo Horizonte – Minas Gerais – Brazil
christ.bh@terra.com.br, lfwgoes@yahoo.com.br, capsms@pucminas.br

Abstract

Consistency is an important issue in Distributed Shared Memory (DSM) systems. These systems share a set of objects or virtual memory pages. The data sharing enables the applications in workloads to access the data concurrently. But, these concurrent accesses can generate some inconsistencies in the shared data state. The consistency models are responsible for managing consistency of shared data for the workloads. In this work, we propose, present and analyze a reconfigurable consistency model for object based DSMs. We called this consistency model ROCoM (Reconfigurable Object Consistency Model). ROCoM behavior was represented using a reconfigurable algorithm (RA) and it analysis was made using a simulation tool (ClusterSim – Cluster Simulation Tool). Our results show that ROCoM, on average, had 55% better performance than the other traditional consistency models.

1. Introduction

In a distributed computing environment, when an application has concurrent access on several shared data, the users must use an abstraction that helps the management of these accesses and simplifies the programming. Distributed shared memory is one of these abstractions that have received considerable attention in the last years [1] [2]. DSM is an abstraction that can be implemented by a specific hardware or software. This abstraction provides an illusion of a shared memory in a parallel and distributed system. Software DSMs are implemented to manage object or virtual memory page concurrent access. In object based DSMs, the operations semantics (Consistency Model) guarantee that objects will be consistent for the process during the workload execution [3] [4] [5].

A consistency model 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 object [2] [6]. The consistency models implementations can be uniform, strong, hybrid and relaxed [1] [2] [4]. The consistency models are composed of independent and well-defined parts (coherence protocols, consistency constraints, events ordering protocols, access policy and replication protocol), and each of one has infinite possible solutions (implementations).

Consistency models should match an easy programming model and reduce the applications response time, in order to maximize the system’s performance for all workloads and architectures. The main problem is that workload, system architecture and consistency model semantics change continuously. In order to solve this problem, some works have been developed to make consistency models more flexible and adaptable [3] [5] [7]. A poorly explored solution is the use of reconfigurable algorithms (RA) in consistency [8] [9] [16] [17].

A reconfigurable algorithm is an algorithm that can change its behavior by altering the form of its configuration. A RA is composed of frames (action and control frames), constructive blocks, structures (wait queues and storage tables) and layers (Basic Layer (BL), Reconfigurable Layer (RL) and Configuration Control Layer (CCL)). Combining these elements, the RA can generate new configurations and reconfigurable itself dynamically. So, this algorithm can be flexible and adapt to the workload and architecture characteristics, improving the performance of the workload execution in architecture [8] [9] [16] [17].

The parts of a consistency model can be combined to form many consistency models. So, the characteristics of reconfigurable algorithms allow the representation of a reconfigurable consistency model. This new consistency model, that we called ROCoM, can change its behavior during the workload execution considering the characteristics of the system.
architecture and workload, giving an ideal performance for each case. ROCoM’s behavior was represented using a reconfigurable algorithm and its analysis was done by means of simulation. So, the ROCoM objectives are: minimize communication time and maximize flexibility, adaptability and performance of the consistency models. According to a deep bibliographic revision [1] [3] [4] [5] [6] [7] [10] [11], we did not find works that apply RA in consistency models proposal, development and implementation.

The main objectives of this paper are: to define and propose the ROCoM, to develop and implement it using a reconfigurable algorithm; to analyze the performance of ROCoM using simulation. The main goal is the implementation and performance analysis of ROCoM in ClusterSim [12] [13].

2. Related Work

In this paper, we will discuss some papers that are more relevant and close to our work [1] [3] [5] [6] [7] [10] [15] [18]. In the paper [7] [18], the authors present some definitions, theorems and proofs that can be used to validate consistency models and its parts. In the works [1] [3] [5] [6] [10] [15], adaptive consistency models are presented.

In [7], Raynal extends the sequential consistency model with WW (Write – Write) and OO (Object conflicting Operations) constraints to histories of multi-objects operations.

In [1], 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.

In [3], 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, the consistency used has to be chosen by the users and not based on the workload and architectures characteristics.

The work [6] proposes a sequential consistency algorithm with dynamic protocols switching. This work also verifies the algorithm by means of formal proofs. In this algorithm, the constraints and the coherence protocols can be changed during the execution. But, just the invalidate lazy coherence protocol is implemented.

The work [10] analyzed the use of reconfigurable algorithms for managing replicated data with strict consistency model. But, they are considering reconfiguration in terms of fault tolerance.

Finally, in [15] the author analyzes the impact of some coherence protocols in a large size DSM. These protocols were analyzed considering operation costs, fault tolerance and scalability. To do the analysis, they simulate the protocols and some workloads that are more used in parallel processing. They conclude with these work that the write invalidate protocol had less communication time than the others.

In the works [3] [5] [6] [7] [18], they present formal proofs of their consistency models, to analyze and validate them. In the other works, [1] [10] [15], experimental and simulation analysis are presented to performance analysis of consistency models and their parts. So, regarding to formal details, the works that are closest to our work are [3] [5] [6] [7] [18] and regarding to experimental results the closest works to our approach are [1] [10] [15].

3. Reconfigurable Object Consistency Model

ROCoM is a reconfigurable object consistency model for asynchronous architectures that execute an object-based software DSM and do not have atomic broadcasting implemented in hardware. This consistency model manages the state of a set of shared objects. ROCoM can have its behavior reconfigured considering the workloads and architectures characteristics. So, it can adapt to the workload and architecture characteristics, becoming more flexible and increasing performance.

As we said before, any consistency model can be decomposed into parts. ROCoM has five parts (an event ordering policy, a constraint policy, a coherence protocol, a replication protocol and an access policy) and an infinite number of implementations that are combined to reconfigurable it. So, it can have its behavior assuming any type of consistency.

The reconfiguration in ROCoM is done during the workload execution, but it is important to say that this reconfiguration is done after an application execution ends. In our system, a workload is composed of \( n \) applications, and between these applications, ROCoM can be reconfigured to assume the best form or configuration to the next application. With this kind of reconfiguration, we minimize the impact of the changes in the consistency model and applications.
In order to not allow inconsistencies during the reconfiguration, ROCoM has a centralized manager (CMan) that ensures that none inconsistency will be generate after the reconfiguration. The CMan controls the set of shared objects and their replicas to do not allow an application to access the objects during the reconfiguration. The rest of the controls, that are necessary for a consistency model, were developed in a distributed way in ROCoM.

The actual version of ROCoM can be reconfigurable to assume atomic or sequential consistency model variations. So, a consistency model is said to be reconfigurable if: 1) it can assume different consistency models during the workload execution time and 2) it can not assume more than one consistency model at a time. Its definition is presented in figure 1.

A reconfigurable algorithm was used to represent ROCoM’s behavior. RA has the characteristics (frames, constructive blocks etc) that were necessary to represent the proposed consistency model. In figure 2, we can observe the RA that represents our proposed consistency model. The CCL of our RA controls the constructive blocks that are active in a moment. It is implemented with a switch case structure. The BL has the implementation of every consistency model’s parts (frames), constructive blocks (possible implementations of consistency models parts) and structures (shared objects and wait queues). Finally, the RL is an instance of the BL.

In this paper the Access Policy Frame can be filled out with single writer/single reader (SWSR) and single writer/multiple reader (SWMR) policies. The Event Ordering Frame is filled out with the sequential and atomic ordering. The Constraint Frame can use the WW or OO approach. The Replication Protocol Frame can be filled out with migration, total and partial replication protocol. Finally, the Coherence Protocol Frame can be filled out with the Update Eager or Invalidate Eager Constructive Blocks.

4. Experimental Environment

In this section, we describe the metrics, distributed architecture and workload used in our simulations. Afterwards, we describe the experimental design in which we highlight the used configurations.

In order to analyze a consistency model, we can use different metrics. The most common are: response time, number of remote instructions, transmitted bytes, coherence messages transmitted time, number of messages and communication time [1] [10] [15].

The mean job response time (in seconds), defined in Eq.1, is the mean time interval between the submission and end of a job.

\[
\text{MeanRespTime} = \frac{\sum_{\text{Job}} (\text{JobEndTime} - \text{JobSubmissionTime})}{\text{NumberOfJobs}} \quad (\text{Eq. 1})
\]

As show in Eq.2, the number of messages is the total number of control messages and object moving messages exchanged between the nodes.

\[
\text{NofMessages} = \text{ControlMessages} + \text{ObjectMessages} \quad (\text{Eq.2})
\]

The communication time is the number of transmitted bytes divided by the network bandwidth, as show in Eq. 3.

\[
\text{CommunicationTime} = \frac{\text{TransmittedBytes}}{\text{NetworkBandWidth}} \quad (\text{Eq. 3})
\]

The number of remote instructions is the total number of instructions that has to send some messages, Eq.4.

\[
\text{NumberOfInstructions} = \text{RReads} + \text{RWrites} \quad (\text{Eq. 4})
\]

As show in Eq.5, the metric transmitted bytes, is the total number of bytes transmitted between the nodes.
The coherence messages transmitted time is the total time spent with the messages of update or invalidate, after a write operation (Eq.6).

\[
\text{CoherenceTime} = \frac{\text{TransmittedCoherenceMessages}}{\text{NetworkBandWidth}} \quad (\text{Eq. 6})
\]

The selected distributed architecture is a cluster composed of 8 nodes interconnected by a Fast Ethernet switch. Each node has a Pentium III 1 GHz processor and an object software DSM. We modeled our environment in ClusterSim, a simulation tool developed by our group [12] [13]. In ClusterSim, a workload is composed of a set of jobs featured by: their types, internal structures, submission probabilities and inter-arrival distributions. To simulate applications that share objects, we have to create the parallel applications defining access operation (write and read) in one or more objects. These objects have also to be created by the user.

DSM applications can be categorized into three broad categories, namely fork-join, run-to-complete and iterative [19]. Each one of these applications has different characteristics. The interactive applications exhibit a regular program behavior and compared with the other two, it is easier to analyze. For these reasons we will simulate different types of them.

In the related works [1] [10] [15], we only found real interactive applications that they used in the experimental tests. So, in order to simulate different applications, we fixed some values and characteristics (number of shared objects, object size and percentage of writes and reads) that are important in an object-based software DSM. So, we create our synthetic applications. These applications represent applications that have just one shared object (Matrix or Vector), like some common problems: Matrix Multiplication, Image Convolution, Quick Short etc [1] [15]. To cover the maximum number of workloads we used percentages (20%, 40% and 60%) to indicate the number of write or read operations in the application. For example, if we had 20% of write operations the other 80% will be read operations. And, we used two objects size (1K bytes-4K bytes) to measure the impact of the object size in the network.

After the choice of the most important workload characteristics, we make some combinations with these characteristics (object size and percentage of write operations) to create our workload, generating 6 applications. The application 1 has object size of 1K and 20% of writes operations; application 2 has object size of 1K and 40% of writes operations; application 3 has object size of 1K and 60% of writes operations; application 4 has object size of 4K and 20% of writes operations; application 5 has object size of 4K and 40% of writes operations and application 6 has object size of 4K and 60% of writes operations. With these characteristics we can generate a large number of different applications and cover the maximum interactive applications space.

<table>
<thead>
<tr>
<th>Table 1. ROCoM configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf.1</td>
</tr>
<tr>
<td>Conf.2</td>
</tr>
<tr>
<td>Conf.3</td>
</tr>
<tr>
<td>Conf.4</td>
</tr>
<tr>
<td>Conf.5</td>
</tr>
<tr>
<td>Conf.6</td>
</tr>
</tbody>
</table>

It is important to note that each ROCoM configuration is a traditional consistency model, in which its parts are fixed and cannot be changed over time. For example, in Table 1, Conf.01 has the WW constraint, Write Update protocol, single writer/multiple reader, total replication protocol and it cannot changes over time. Through the rest of this paper, a traditional consistency model and configuration will be treated as synonyms.

In order to test and analyze the performance of the ROCoM, we used some combinations of a consistency model parts. A configuration of ROCoM is composed of an Access Policy, an Event Ordering Policy, a Constraint, a Coherence Protocol and a Replication Protocol. In this implementation of ROCoM, the event ordering is fixed in sequential and atomic, because we are comparing our ROCoM with traditional sequential and atomic consistency models implementations. So, considering the implemented constructive blocks, ROCoM has 6 configurations as shown in Table 1. We made a total of 36 (6 configs X 6 apps) simulations.

5. Results

For each metric, in the performance analysis of ROCoM, we present and analyze the results obtained with each form or configuration of ROCoM. Each ROCoM’s configuration is a traditional implementation of a uniform and strong consistency model (Sequential or Atomic). So, to analyze the performance of ROCoM we compare it to each traditional consistency model implementation. We also present the results for each application on the tested workload.
5.1. Configurations Analysis

In figures 3, 4 and 5, we present the relative means of transmitted bytes, coherence time and communication time among each configuration for the simulated workload. We can observe in these figures that the results growth is exponential. This happens because the number of writes and the size of object increase for appl.1 to appl.6, increasing the amount of communication and bytes between the nodes.

Figure 3. Relative means Transmitted Bytes among each configuration for workload

In the graphics presented in figures 3 and 4, we can see that Conf.05 presented better results than the Conf.02 (better on average for the others applications) for the appl.6. In Conf.02 after a write operation, an invalidate message was sent to every node that has an object replica. Furthermore, when the nodes receive the message sent, its object replica is invalidated. After the invalidation, the application has to request the object in the next operation on the object. So, the Conf.02 sent n x (invalidate msg + object msg + request msg), where n is the number of nodes that has the object replica at that moment, while Conf.05 sent n x (object msg + request msg). Appl.6 has a large number of write operations and an object size of 4Kbytes. So, the invalidate coherence protocol is not a good choice for this application. In the others applications, Conf.02 was better than the Conf.05, because the objects size and the number of writes are smaller than in appl. 6.

Figure 4. Relative means Communication Time among each configuration for workload

The coherence time metric measures the time spent with the coherence protocols. As we can see in figure 5, the Conf.02 was best to the appl.1 and the Conf.05 was the best to the others five applications. In these configurations, the coherence protocol that was used is the update, not the update eager. This protocol updates just one object replica, the one that moves between the nodes, while the update eager updates all object replicas. So, the total coherence time of the update protocol is the number of writes operations that were made by the application, while in the others protocols the number of messages is multiplied by the number of replicas, increasing the time.

Another important result about this metric is that the invalidate eager presented smaller coherence time than the update eager. This happens because, the invalidate messages are at least 10 times smaller than the update messages.

Figure 5. Relative means Coherence Time among each configuration for workload

In figure 6, we present the relative mean number of remote instructions among each configuration for all applications on the simulated workload. We can see, in this figure, that configurations 3 and 6 presented the same results for all applications.

Figure 6. Relative means Remote Instructions among each configuration for workload

These configurations also presented the best performance for the simulated workload. Conf.03 and Conf.06 have the update eager protocol and the total replication protocol. When these two protocols are used in conjunction we only have local instructions, because, all nodes have the object replica and the update eager protocol will never invalidate the object.
replicas. So, for this metric, the configurations 3 and 6 had none remote operations, becoming the best configurations for all applications.

As we can see in figure 7, the Conf.06 presents the best performance for the metric response time for all applications. This configuration has total replication protocol and WW constraint. With total replication protocol, this configuration can allow fast writes and some fast reads. A fast operation happens when it is executed locally and has smaller response time. Furthermore, with WW constraint, this configuration restricts less concurrent operations, just the concurrent writes, allowing more operations to be executed in parallel. So, regarding to response time, we can conclude that the Conf.06 is better for all applications.

Figure 7. Relative means Response Time among each configuration for workload

In figure 8, we can see that Conf.02 and Conf.05 presented the best results for the metric number of messages. For this metric, the consistency model parts that have more influence are replication protocol and the coherence protocol. As Conf.02 uses partial replication, the number of messages sent through the network is smaller than if it uses total replication. In the partial replication protocol, every time that application requests an object, its node receives an object replica. So, the application could have the object locally, not having to request the object again in the next operation, like in the migration replication protocol. But, this local object replica does not eliminate the messages on the network. Because, when we use partial or total replication after a write operation, we always have to send an update or an invalidate message. In partial replication, the objects are partially replicated in the nodes, while in the total replication, the objects are totally replicated in the nodes. So, the number of messages in the partial replication protocol is smaller than in the total replication protocol.

When we have applications with a large number of writes is better to use the migration replication protocol, like in Conf.05, because with this replication protocol we have not to use the invalidate or update eager coherence protocol. In a migration protocol, the object replica moves between the nodes when the applications request. So, the number of sent messages is equal to the number of requested objects. It is important to say that the number of messages sent increases just when the number of writes and remote operation increases. For the metric number of messages, we can conclude that the object size does not change the results. We can also conclude that, for applications with large number of writes (bigger than 40%), the best configuration is Conf.05, and for applications with smaller number of writes (smaller than 20%), the best configuration is Conf.02.

5.2. ROCoM Analysis

In order to analyze the performance of ROCoM, we need to compare it to each configuration or traditional consistency model individually. As we said in the proposal of ROCoM, the CCL evaluates the entry parameters, reconfiguring ROCoM to the best configuration. As we are not using a real trace, we analyze ROCoM for all applications.

In Table 2, we observe that on average, considering all metrics, ROCoM is 55.99% better than the 6 traditional consistency models. Note that if we had chosen Conf.02 (the best configuration on average) to be the consistency model of our system, ROCoM would still be 39.57% better.

As an example, suppose that we had a system with a traditional sequential consistency model like Conf.06. If we had chosen response time as the metrics in our CCL algorithm, our ROCoM will take 0% of speedup in relation with the Conf.06. However, if we had chosen transmitted bytes or communication time as a
metric for the CCL, our ROCoM reach 80% of speedup.

Now, if we had considered Conf.04 as the traditional consistency model of our system, it would reach the worst results for the simulated workload. This configuration uses the update eager coherence protocol, and this protocol is good just when we have a larger number of writes (higher than 60%) with a small number of reads at the same object (not present in our workload). In this case, the read operations will need the object new value and the update protocol (object message) will give it to the reads operations with a smaller number of messages than the invalidate protocol (invalidate message and object message). As we showed in our results, ROCoM never assumed this configuration, because Conf.04 had the worst performance and ROCoM always assume the best configuration for an application and architecture.

Table 2. Speedup, in percentage (%), of the ROCoM performance when compared to each configuration

<table>
<thead>
<tr>
<th>Metric</th>
<th>Transaction Time</th>
<th>Communication Time</th>
<th>Number of Messages</th>
<th>Invalidate Instructions</th>
<th>Transmitted Bytes</th>
<th>Coherence Time</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conf.01</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.02</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.03</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.04</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.05</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.06</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.07</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.08</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
<tr>
<td>Conf.09</td>
<td>105.05</td>
<td>45.14</td>
<td>22.32</td>
<td>118.85</td>
<td>36.08</td>
<td>28.84</td>
<td>54.57</td>
</tr>
</tbody>
</table>

With these examples, we show that the use of reconfiguration in consistency models may provide a high speedup over traditional consistency models. In these examples, we considered that there weren’t reconfiguration overheads and wrong workload classifications. The reconfiguration overhead is insignificant, it is just the time spent to execute a switch case and fit some specific blocks in frames to change the configuration. In despite of these overheads and costs, the speedup may be great enough to make ROCoM a good alternative.

6. Conclusions

In this paper, we proposed, developed, implemented and analyzed the performance of ROCoM by simulation. To propose this model, we use formal consistency models definitions and to develop, we used a reconfigurable algorithm. As general conclusions about the ROCoM frames we can highlight:

Event Order Frame. Considering all metrics, on average, sequential event order presented the best equivalent performance, as in [4] [14]. It suggests that an atomic constructive block is not a good choice to be implemented in an object based DSM.

Constraint Frame. About the constructive blocks of this frame, we can say that WW constraint blocks a smaller number of concurrent operations. WW constraint blocks just when the applications have two or more concurrent write. On the other hand, the OO constraint blocks two or more concurrent writes and conflicting operations (read and write in the same object). So, if we considering response time or remote instructions, the WW constraint will present better performance, because there is more possibility of parallel operations execution.

Coherence Protocol Frame. Considering the metrics communication time, coherence time and transmitted bytes, the invalidate constructive block presented the best performance, as show in [2] [15] [11]. In this constructive block, the invalidate message size is just 96 bytes, while in the update constructive block this sizes increases to the object size (1K – 4K).

Replication Protocol Frame. Considering the metrics communication time, coherence time and transmitted bytes, the invalidate constructive block presents the best results, because just one object will be moving between the nodes. On the other hand, if we are interesting in faster operations (local operations), fault tolerance and availability the total and partial constructive blocks are better. Because, we have more replicas of the same object. But, these replicas increase the number of control messages and object messages and decrease the performance.

Access Policy Frame. We can say that if we use the multiple readers constructive blocks with the atomic event order its performance decreases. Because the atomic event order will reduce the possibility of parallel read operations. So, in our simulations, the single reader had better performance than the multiple readers, because it was used with the sequential event order that is more relaxed than the atomic.

On average, the performance of ROCoM was 55% better than the other traditional consistency models for the tested workload. One of the most important results was to show that depending on the selected metric (Response time and remote instructions), the best algorithm on average (Conf.02) may be worse than one of the worst consistency models on average (Conf.06).

In this paper, we proposed ROCoM, a
reconfigurable consistency model for objects that can be configured to be some variations of atomic and sequential consistency models. With the five frames describe in this paper, developers can add other blocks and do other consistency models, like the relaxed one that generates, on average, better performance.

The main contributions of this paper are: the proposal, implementation and performance analysis of ROCoM, comparing it with other traditional consistency models for different applications.

As future works we can highlight: the inclusion of new blocks in ROCoM; an adaptive CCL; study on how to classify the workload; tests with other workloads, simulations of many instructions and nodes, and tests in a real environment.

7. References


