An Efficient and Scalable Approach for Implementing Fault Tolerant DSM Architectures

Christine Morin, Anne-Marie Kermarrec, Michel Banâtre

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An Efficient and Scalable Approach for Implementing Fault Tolerant DSM Architectures

Christine Morin, Anne-Marie Kermarrec, Michel Banâtre

N ° 3103

Février 1997

THÈME 1

ISSN 0249-6399

An Efficient and Scalable Approach for Implementing Fault Tolerant DSM Architectures

Christine Morin*, Anne-Marie Kermarrec†, Michel Banâtre‡

Thème 1 — Réseaux et systèmes
Projet Solidor

Rapport de recherche n ° 3103 — Février 1997 — 40 pages

Abstract: Distributed Shared Memory (DSM) architectures are attractive to execute high performance parallel applications. Made up of a large number of components, these architectures have however a high probability of failure. We propose a protocol to tolerate node failures in two classes of DSM architectures: Cache Only Memory Architectures (COMA) and Distributed Virtual Shared Memory (SVM) systems. The proposed solution is based on backward error recovery and consists of an extension to the existing coherence protocols to manage data used by processors for the computation and recovery data, used for fault tolerance. The implementation of the protocol in a COMA architecture has been evaluated by simulation. The protocol has also been implemented in a SVM system on a network of workstations. Both simulation results and measurements show that our solution is efficient and scalable.

Key-words: Distributed Shared Memory, Fault Tolerance, Coherence Protocol, Backward Error Recovery, Scalability, Performance.

(Résumé : tsvp)

The work presented in this paper is partially funded by the DRET research contract number 93.34.124.00.470.75.01.

* cmorin@irisa.fr
† amk@cs.vu.nl
‡ banatre@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 02 99 84 71 00 – Télécopie : (33) 02 99 84 71 71

Une approche efficace et extensible pour mettre en œuvre des architectures à mémoire partagée répartie tolérantes aux fautes

Résumé : Les architectures à mémoire partagée répartie sont attrayantes pour l'exécution d'applications parallèles à haute performance. Composées d'un grand nombre d'éléments, ces architectures ont cependant une probabilité de défaillance très élevée. Nous proposons un protocole reposant sur le retour arrière, pour tolérer les défaillances des nœuds dans deux classes d'architectures à mémoire partagée répartie : les *Cache Only Memory Architectures* (COMA) et les systèmes à mémoire virtuelle partagée (MVP). Sa conception repose sur l'exploitation des caractéristiques des architectures considérées qui permet de stocker les données de récupération, celles restaurées en cas de retour arrière, en mémoire vive assurant un etablissement et une restauration rapide de points de récupération tout en évitant le développement de matériel spécifique coûteux. La solution proposée est fondée sur un protocole de récupération arrière et consiste en une extension du protocole de cohérence pour gérer à la fois les données utilisées par les processeurs et les données de récupération. L'impact de l'utilisation du protocole dans une architecture COMA a été évalué par simulation. Le protocole a également été mis en œuvre au sein d'une mémoire virtuelle partagée sur un réseau de stations de travail. Les résultats de simulation et les mesures montrent que notre solution, outre le fait d'être générique, est à la fois efficace et extensible.

Mots-clé : mémoire partagée répartie, tolérance aux fautes, protocole de cohérence, retour arrière, extensibilité, performance.

1 Introduction

Distributed Shared Memory (DSM) architectures are attractive for the execution of high performance parallel applications since they provide the simple shared memory paradigm in a scalable way. Such architectures rely on the automatic migration and replication of data so that the data is local to the processors using it for computation, in order to execute parallel applications at maximum speed. Scalable shared memory architectures and software-based DSM implemented on multicomputers or networks of workstations (NOW) are examples of such architectures. These architectures implement a coherence protocol to manage the multiple copies of a data in different node memories. Due to their large number of components, and despite a significant increase in hardware reliability, these architectures may experience hardware failures. Thus, tolerating node failures becomes essential if such architectures are to execute long-running applications whose execution time is longer than the architecture Mean Time Between Failures (MTBF). Backward Error Recovery (BER) [1] is part of a fault tolerance strategy which restores a previous consistent system state after a failure occurred. To achieve this goal, a consistent system state, made up of a set of recovery data, has to be periodically saved on stable storage. A stable storage ensures that (1) data is not altered (*permanence* property) and remains accessible (*accessibility* property) despite a failure, and that (2) data is updated atomically in presence of failures (*atomicity* property). BER seems to be the more attractive solution for providing fault tolerance in DSM architectures since it limits the hardware cost compared to approaches based on static hardware replication. In contrast with active replication schemes, it allows the use of all the processors for computation.

In this paper, we propose a low overhead and scalable error recovery approach based on BER to tolerating transient and single permanent node failures in the class of DSM architectures exemplified by Cache Only Memory Architectures (COMA) and Shared Virtual Memory (SVM) architectures. We have simply extended the standard coherence protocol of a DSM. The extended coherence protocol (ECP) is designed to manage both data accessed by processors for computation and that required for recovery. Our approach has lots of advantages. First, both types of data are stored in the node standard memories so no specific hardware needs to be developed. Implementing the protocol in a COMA only requires slight modifications to existing cache controller, but no new functionality needs to be added. Another significant advantage of our scheme is that, on one hand, it takes benefit of the replication inherent to the DSM by using data already replicated to avoid the need to create additional recovery data. On the other hand, data may be used until it has been

modified after a recovery point. Thus, the replication required for providing fault-tolerance can be exploited during failure-free executions.

Performance results obtained by simulation in COMAs and measured in SVM architectures implementing the proposed protocol show that our approach is both efficient and scalable. It is efficient as it limits the disturbance caused by the fault-tolerance on failure-free execution. Our approach is also both generic and scalable. As the number of memory modules grows with the number of nodes in COMAs and NOWs, storing recovery data in standard memories does not limit their scalability. Moreover, the low latency high bandwidth network of such architectures allows a large number of processors to be used efficiently.

The remainder of this paper is organized as follows. Section 2 presents a generic model of DSM systems from which two architecture classes can be derived: COMA and SVM. In Section 3, we first explain BER requirements and then describe the extended coherence protocol we propose to implement an efficient BER strategy in DSM architectures. Issues raised by the implementation of the coherence protocol in COMA and SVM architectures are discussed in Section 4. Results of the performance evaluation for the considered architectures are provided in Section 5 where we also point out the advantages of our approach. Section 6 concludes.

2 Distributed Shared Memory Architectures

2.1 Model

DSM systems are composed of a set of processing nodes interconnected though a scalable interconnection network. Each node is composed of one or more processors, their associated caches and a memory. A DSM system provides a single shared address space despite distributed memories. Thus, it is particularly attractive from the programmers' point of view since it provides a simple way of programming. Moreover, distributing the memory among processing nodes ensures the scalability of the system. Scalable architectures provide possibility of increasing computing power.

There are two classes of DSM architectures: those which implement a static physical address space like *Cache Coherent Non Uniform Memory Access* (CC-NUMA) architectures such as DASH [2] or the Sequent STiNG architecture [3], and those which provide a dynamic address space like COMA such as DDM [4] or the KSR1 machine [5]. In the former class, each memory represents a static portion of the global address space, whereas in the latter, the memory of a node is used as a large cache for the node's processors.

In this paper, we focus on the architectures implementing a dynamic address space, for which a generic model is depicted in Figure 1. Both COMAs, in which the shared address space is implemented by specialized hardware, and SVM systems, which are implemented by software on multicomputers such as the Intel Paragon [6] or NOW [7] are based on a dynamic address space. In such systems, data is automatically replicated or migrated on demand to the memory of the node of the processor that wishes to use the data in a computation, in order to exploit both spatial and temporal locality and decrease memory latencies. These transfers are managed transparently from the application's point of view.

The main characteristics of such architectures are that first there is no fixed physical location for data and second, since replication of an item is allowed in several memory modules for efficiency, a coherence protocol is required to maintain consistency between the multiple copies. In write-update protocols, on each write on a memory block, all the nodes having a copy of the same block also perform the update. In contrast, in write-invalidate protocols, an invalidation of all existing copies of a replicated item is required when it is modified by a processor. Write-invalidate coherence protocols [8] are preferred to write-update ones [9] since a coherence operation is not required at each write. We describe such a protocol at the end of this section. Two kinds of implementation of write-invalidate protocols exist: snooping and directory-based. Snooping protocols imply that all memories see every memory requests. In directory-based protocols, a directory contains one entry for each memory block, maintaining the list of nodes holding a copy of the block. When a miss occurs, the directory is first read to forward the request to the node that is able to deal with it. Directories may be static or dynamic. In the former case, each node manages a portion of the directory, which is statically defined. In the latter, a directory entry has no static location but is associated with the current owner of the block. Directory-based protocols are preferred in DSM systems since they do not compromise scalability of the system.

A standard write-invalidate protocol is shown by the transition diagram in Figure 2. We assume the concept of unique ownership [8], the owner of the block being the unique processor allowed to serve write misses. A memory block b of the shared address space may be in one of the following states :

- *Modified exclusive* (ME): the memory of node n contains the unique copy of block b. It can be read or written.
- *Non exclusive owner* (NEO): the memory of n contains a copy of b which can only be read. Other copies may exist in the system in state *Shared*.