ICARE: COMBINING EFFICIENCY AND HIGH-AVAILABILITY IN A DSM SYSTEM

ANNE-MARIE KERMARREC
ICARE: Combining Efficiency and High-Availability in a DSM System

Anne-Marie Kermarrec

Thème 1 — Réseaux et systèmes
Projet Solidor

Publication interne n° 1077 — Janvier 1997 — 23 pages

Abstract: In light of the increasing throughput of local area networks, Networks Of Workstations (NOW) which provide a distributed shared memory (DSM) have become a convenient alternative to parallel architectures in the framework of parallel scientific applications. ICARE is a recoverable DSM based on backward error recovery which is implemented on top of an experiments ATM platform running the CHORUS microkernel. ICARE provides a way to combine both efficiency and high availability in such a system. The fact that recovery data are stored in standard memory provides a low-cost fault-tolerance mechanism as well as the opportunity to exploit the symbiotic relationship between the data replication existing in essence in DSM systems and the one needed for fault-tolerance to increasing efficiency. Furthermore, ICARE implements an efficient transparent rollback recovery. Indeed, the disturbance of a rollback is minimized by distributing the load in a balanced way among the remaining fault-free nodes in the event of a permanent failure. Performance evaluations show the efficiency of the ICARE prototype that implements the proposed algorithms.

Key-words: Network of Workstations, Distributed Shared Memory, Backward Error Recovery, Data Replication, Efficiency

(Résumé : tsyp)
ICARE : une mémoire virtuelle partagée alliant efficacité et tolérance aux fautes

Résumé : À l’heure de l’avènement des réseaux locaux haut débit, les réseaux de stations de travail qui mettent en œuvre une mémoire virtuelle partagée (MVP) représentent une alternative intéressante aux multiprocesseurs à mémoire partagée pour l’exécution d’applications parallèles. ICARE est une MVP recouvrable mise en œuvre sur une plate-forme d’expérimentation reposant sur ATM et utilisant le micro-noyau CHORUS. ICARE conjugue la tolérance aux fautes et l’efficacité en stockant les données de récupération en mémoire vive. Ceci lui permet d’exploiter la synergie observée entre la réplication des données, inhérente au fonctionnement d’une MVP et celle indispensable à la tolérance aux fautes. En outre, ICARE met en œuvre une reprise transparente et efficace de l’application en cas de défaillance en distribuant de manière uniforme la charge du site défaillant. Les évaluations de performance menées à partir d’un prototype montrent l’efficacité des algorithmes proposés.

Mots-clé : Réseaux de stations de travail, mémoire virtuelle partagée, retour arrière, réplication de données, efficacité
1 Introduction

With the emergence of high-speed local area networks, Networks Of Workstations (NOW) [2] providing distributed shared memory [1], are an attractive alternative to shared memory multiprocessors for the execution of parallel applications. DSM systems such as NOWs offer a potential scalability in combination with an intuitive way of programming. However, made up of a large number of components, NOWs are exposed to experience hardware failures as well as node reboot despite the increasing reliability of hardware. Backward Error Recovery (BER) is a fault-tolerant scheme well suited to parallel applications that are not subject to real-time constraints. It consists of periodically saving a consistent snapshot of the entire system (called a recovery point or a checkpoint) on a stable storage [8], which ensures that recovery data survive failures, and to restore this snapshot when a failure is detected.

ICARE is a recoverable DSM designed and implemented on a NOW to provide fault tolerance in an efficient way. It tolerates multiple transient failures or a single permanent one. The proposed DSM system exploits the synergy between the replication inherent to a DSM and the one needed in a BER scheme. The efficiency of ICARE relies on the three following features: (i) it exploits the already existing mechanism of a DSM system to limit the impact of a BER implementation, (ii) it exploits the data replication needed in a BER scheme to increase the efficiency of the DSM system and (iii) it ensures a transparent and efficient rollback recovery when a permanent failure occurs.

First of all, ICARE exploits the fact that two memory modules are failure-independent in a NOW and stores recovery data\(^1\) in standard memory. In the absence of dedicated reliable hardware to implement stable storage, permanence of recovery data is guaranteed by replicating them in two distinct main memories. Such an approach ensures a fast establishment of, and restoration from, a recovery point. Indeed, it does not suffer from the high access latency of disks [6, 18]. Moreover, it provides a low-cost solution by avoiding the need for dedicated hardware. Second, as recovery data are stored as active ones in standard memories, their management is ensured by the coherence protocol extended for this purpose. This method has been initially introduced for a Cache Only Memory Architecture [10]. Finally, ICARE benefits from the data replication implemented in a DSM to decrease latency of re-

---

\(^1\)We distinguish recovery data, which belong to a recovery point, from active ones, which are those used by processors for the computation.
mote accesses and to exploit locality of references, in order to avoid the creation of recovery data.

Conversely, ICARE exploits the replication induced by the periodic establishment of recovery points to avoid page faults which would have been arisen in a DSM without fault-tolerance. This feature increases the overall efficiency of a DSM system since handling page faults in a DSM system is an expensive operation which generates a lot of messages to locate referenced data, to maintain coherence and to transfer data. Using recovery data replicated for achieving fault-tolerance during the computation thus avoiding page faults is possible in ICARE since recovery data are stored in standard memory and remain readable by processors until modified. The fact that BER requires replication leads us to expect a potential performance gain by further exploiting this replication to avoid page faults during standard execution. To attain these favorable situations, we create memory affinity, by controlling data replication of recovery data at checkpointing time, in order to store pages close to processes using that data for computation.

The last attractive feature of ICARE that we mention there, is that it ensures a transparent and efficient rollback when a permanent failure occurs. ICARE ensures memory reconfiguration to allow the system to tolerate another failure immediately after the rollback. Moreover, it distributes the load of the faulty node (computation and manager loads) in an uniform way among the remaining fault-free nodes.

This paper is organized as follows. Section 2 describes the framework of the study. Section 3 describes the design of ICARE and the checkpointing algorithm. Two variants of this algorithm are proposed that increase efficiency of the whole system. The recovery of ICARE in a uniform way after a permanent failure is presented in Section 4. Section 5 demonstrates the efficiency of ICARE by showing the results of experiments which come from the implementation of a prototype on a NOW. Section 6 compares our work to that of others.

2 Framework

2.1 Model of Computation

We consider a NOW consisting of a collection of nodes connected by ATM, with each node composed of a processor and a memory module. The basic DSM offers the abstraction of a shared virtual memory on top of the distributed memory
modules where the *page* is used as the unit of sharing and transfer between processors. A page may be replicated across several node memories and a coherence protocol is required to manage consistency between replicas. ICARE is designed as an extension of the basic DSM scheme devised by Li *et al.* [9] i.e which implements a write-invalidate statically distributed directories based protocol. Ownership is managed per page. We distinguish the *owner* of a page from its *manager*. In a system with *N* nodes, the manager $M(p)$ of a page $p$ is statically defined by function $M(p) = p \mod N$. A manager contains a directory entry for each page it is responsible for. The owner owns an up-to-date copy of the page and manages its copyset, which is a list of nodes that have a copy of the page. A page may be in one of the following states: *invalid* (INV), *modified exclusive* (MOD-EX) when the page is unique and writable, *non exclusive owner* (NEO) when the page is readable, and *shared* (SH) when the page is readable and another copy already exists (at least a NEO one). MOD-EX and NEO states refer to the ownership.

### 2.2 Fault-Tolerance Features

The local-area network is assumed to provide reliable communication between nodes. The nodes are assumed to adhere to *fail-stop* behavior, namely the failure of a node element does not alter fault-free nodes. A node component failure leads to the unavailability of the whole node.

Application-transparent error recovery is achieved in ICARE through a BER scheme [8]. A coherent state of the entire system (recovery point) is periodically saved on stable storage and restored if a failure is detected. Error detection and confinement are out of the scope of this paper. We consider an incremental global coordinated checkpointing policy [3] where all nodes establish simultaneously a recovery point. Such a scheme minimizes the overhead during normal operation and avoids the need to track dependency information between nodes while avoiding the domino effect [12].
3 Providing High Availability and Efficiency in ICARE

3.1 Principles

ICARE relies on the exploitation of DSM features. First, memory modules, which are failure independent in a NOW, are used to store recovery data, thus ensuring a fast recovery point establishment where both the network high bandwidth and memory throughput are exploited. Second, as pages have no fixed physical location in a DSM, they may be stored and restored anywhere in the system, thus greatly simplifying fault-tolerance. Finally, data replicated on account of DSM alone are simultaneously used as recovery data, thus limiting the cost of a recovery point establishment by avoiding the creation of recovery data and requiring no extra memory for recovery data. Conversely, DSM efficiency may benefit from the data replication required for fault tolerance. Recovery data remain readable by processors as long as they have not been modified since the last checkpoint. So they can also be used for anticipating page faults during failure-free executions.

The coherence protocol of ICARE is extended to combine recovery and active data management. In order to identify recovery data and distinguish them from active data, two additional states are introduced in the ICARE coherence protocol: shared-checkpoint (SH-CK) and invalid-checkpoint (INV-CK). SH-CK state refers to pages belonging to the recovery point that have not been modified since that point. Such pages can still be accessed by the processors as they are identical to active pages. So they are still readable and may serve page faults. INV-CK state refers to pages which belong only to the recovery point and will be used only in the event of a failure. The creation of INV-CK pages is delayed until the first modification on a SH-CK page after the establishment of a recovery point. The two recovery copies of a page are numbered and the number one refers to ownership.

3.2 Establishing a Recovery Point

To establish a recovery point on a node, each node creates for every page it owns (pages MOD-EX or NEO pages), two recovery copies on two distinct memory modules. Two cases may arise, depending on the state of a page and the number of copies which exist.
• **Single copy**: such a page is used by only one node in state MOD-EX or NEO. The local page becomes a recovery copy (state SH-CK1) while a second copy (state SH-CK2) is created on some other node.

• **Multiple copies**: such a page is in NEO state and other SH pages exist. The local page becomes a recovery copy (state SH-CK1) while one of the already existing copies is turned into the second recovery copy (state SH-CK2).

At the establishment of a recovery point, INV-CK pages, which belong to the old recovery point, are discarded, while SH-CK pages which belong to both the old and the new recovery point are left unchanged. SH copies do not belong to any recovery point and remain unchanged.

Processes also need to be checkpointed. For that purpose, every node saves locally a checkpoint of the context of each process and replicates its context on another node, according to the same two-phase commit protocol. The data which refer to the context of processes are called private data.

### 3.3 Writable Pages Optimization

The basic checkpointing algorithm of ICARE allows the reading of recovery pages as long as they have not been modified. However, one of its major drawbacks is that it transforms writable pages (MOD-EX) in readable ones (SH-CK). This feature increases the number of access rights growths \(^2\) when subsequent writes on a SH-CK page arise, and may have a significant impact on the overall performance.

Consider an application that performs only writes. At a recovery-point establishment, writable pages become readable. A subsequent write on a SH-CK page leads, apart from the creation of a recovery INV-CK page locally, to the invalidation of the remote SH-CK copy and its change to INV-CK state. These expensive operations can be avoided if the treatment differs between readable and writable pages at the establishment of a recovery point.

The writable page optimization (WPO) consists of creating recovery pages which correspond to MOD-EX pages, in INV-CK state. We thus disallow a processor access to such a recovery page. Then, the MOD-EX pages are kept in MOD-EX and remain writable. The two corresponding recovery copies are directly created in INV-CK

---

\(^2\)An access rights growth arises when a page is referenced through a write operation whereas it is just readable. At worst, such an operation may imply invalidations.
state, one locally and the other one on a remote node. The treatment remains unchanged for NEO pages. This algorithm is offered to programmers who know the memory access patterns of their applications.

3.4 Creating Memory Affinity for Efficiency

```
begin // executed on each node
{for every page p the node owns
{p.state =SH-CK1;
// local recovery copy
	case (p.state)
	single_page:// MOD-EXC or unique NEO page
	{
		{if p.histolist !={}\// the histolist is not empty
			node = p.histolist
		// the first node of the list is chosen
		else
			node = (node + 1) mod (nb_nodes)
	// the default node is the neighbor
	endif
	replication(node, p);
// creation of sh-ck2 copy on node
	}

multiple_page:
// NEO page, at least another SH copy exists in the system
	{node = p.copysetlist-elem;
// the first node belonging to the copysetlist is chosen
	exploitation(node,p),
	}

shared,CK: skip;
invalid,CK:

Default: skip;
}}
@end
```

Figure 1: Establishment of a recovery point according to the history-based policy

One of the main advantage of our checkpointing algorithm is that it allows processors to read recovery pages as long as they have not been modified. It thus allows the use of recovery pages which have been replicated for reasons of fault-tolerance. In order to increase the use of recovery pages, thus increasing the DSM efficiency, we propose to control the data replication at the establishment of a recovery point in a judicious fashion. The primary goal is to transform expensive page faults in less expensive DSM operations such as access rights growths or ideally, hits. For that purpose, we propose to create memory affinity between processes and node memories. In particular, we wish to anticipate page faults using the property that parallel applications present locality in their memory accesses. Memory affinity is derived from the cache affinity concept [16] used in shared memory multiproces-
sors, to schedule a preempted process on a processor whose cache still contains a part of the process’ working set.

As replication is necessary in ICARE and because of the temporal locality principle [5], we propose to create recovery data on nodes which have used the pages recently and thus have a high probability to reference them in a near future. The history of a page is maintained by recording, the identity of nodes which have accessed the page recently in the page descriptor field histolist. If a node \( i \) references through a write operation a page \( p \) owned by the node \( j \) (\( p \) ownership is transferred from \( j \) to \( i \)), the histolist is built up by setting: \( histolist_i(p) = copyset_j(p) . \text{first} \) or \( j \). Indeed, if the copysetlist is empty, the only available information about the recent history of the page is given by the previous owner, otherwise the first element of the copysetlist is chosen. Figure 1 presents the establishment of a recovery point according to the history-based policy. The algorithm creates memory-affinity by using the histolist since recovery copies are created on nodes which will probably reference them in the future thereby anticipating page faults. Information needed to construct the histolist (namely copysetlist and previous owner) is inexpensive to collect since it is already available for implementing the write-invalidate coherence protocol of the basic DSM.

### 3.5 Summary

Table 1 summarizes the pages state changes induced by each of the three recovery point establishment proposed algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Basic Algorithm</th>
<th>History-based Algorithm</th>
<th>WPO Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modified exclusive</td>
<td>Creation of a shared-CK2 page on the neighbor node</td>
<td>Creation of a shared-CK2 page on histolist node</td>
<td>Creation of two Invalid-CK pages, one locally and one remotely</td>
</tr>
<tr>
<td>Non Exclusive Owner</td>
<td>Creation of a shared-CK2 page on the neighbor node</td>
<td>Creation of a shared-CK2 page on histolist node</td>
<td>Creation of a shared-CK2 page remotely</td>
</tr>
<tr>
<td>Shared</td>
<td>Creation of a shared-CK2 page on the neighbor node</td>
<td>Creation of a shared-CK2 page on histolist node</td>
<td>Creation of a shared-CK2 page remotely</td>
</tr>
<tr>
<td>Shared-CK</td>
<td>• Shared-CK1 if chosen page, unchanged otherwise</td>
<td>• Unchanged</td>
<td>• Invalid</td>
</tr>
<tr>
<td>Invalid-CK</td>
<td>• Invalid</td>
<td>• Invalid</td>
<td>• Invalid</td>
</tr>
</tbody>
</table>

Table 1: Summary of page state changes for the three proposed recovery point establishment algorithm.
3.6 Restoring a Recovery Point

When a failure is detected, the previous recovery point, which refers both to shared and private data, is restored. The shared memory recovery point is made up of the set of SH-CK and INV-CK pages. Each node scans its memory and restores all INV-CK pages in SH-CK ones while invalidating all other pages according to the algorithm depicted on Figure 2. In the event of a transient failure, processes states are restored locally from the previous checkpoint and the system can restart. When a permanent failure is detected, a memory reconfiguration must also be performed as a memory module’s entire content has been lost. Moreover, processes which were executed on the faulty node have to be restarted on a non-faulty one. Such a reconfiguration is described in Section 4.

begin // Rollback Recovery
{ for every page p in memory
{ case(p.state) in
    Invalide-CK:
        p.state = Shared-CK;
    Shared-CK: skip
    Shared:
        Modified-exclusif:
            Non Exclusif Owner:
                p.state = invalid:
{ endcase
}endfor}end

Figure 2: Rollback algorithm

4 Permanent Failure Recovery

A permanent node failure leads to the loss of the manager of a static set of pages, the loss of a memory module’s entire content and thus recovery pages, the potential loss of the owner of a set of pages, and finally the loss of a computation unit (Figure 3). These three losses are discussed in the following subsections.
4.1 Sparre Manager Function

In the event of a permanent failure, pages which were managed by the faulty node now become orphan. In order to keep the manager load well-balanced in the system, it is not desirable to assign this whole set of orphan pages to the same node. To avoid such an inefficient scheme, we propose a static spare manager (SM) function which ensures a uniform reassignment of the manager load among the remaining fault-free nodes.

We consider a system with $N$ nodes, numbered from 0 to $N - 1$. We then consider the failure of node $y$, initial manager of the ordered list of pages $P$, such as $p_k = y + kN$, $k \in \{0,...,P - 1\}$. After a permanent failure, the system is then composed of $N - 1$ nodes but the numbering remains the initial one. We want to distribute in an uniform way the $P$ pages managed by $y$ on the $N - 1$ remaining fault-free nodes.

The spare manager function must establish a one-to-one mapping between $\{0, 1, 2...y - 1, y + 1,...N - 2\}$ and $\{0, 1, 2...y - 1, y + 1,...N - 1\}$. Moreover it must be a function of $k \mod (N - 1)$ to ensure the uniform redistribution. So, $SM(p_k) = f[k \mod (N - 1)]$ with

$$f : \{0, 1, 2..., N - 2\} \rightarrow \{0, 1, 2..., y - 1, y + 1,...N - 1\}$$

$$k \mod (N - 1) \mapsto f[k \mod (N - 1)]$$

Figure 3: Impact of a node permanent failure
That is to say \( f(x) = (x + y + 1) \mod N \). The spare manager \( SM(p) \) of a page \( p \) is then obtained by the following uniform redistribution function:

\[
SM(p) = [k \mod (N - 1) + M(p) + 1] \mod N
\]

where \( p = M(p) + kN \), is the address of the considered page, \( M(p) \in \{0, ..., N - 1\} \) is the faulty node and the initial manager of \( p \), and \( k = \frac{M(p)}{N} \) is the ordering number of the considered page in the page list initially managed by \( M(p) \). The function \( f \) is then applied to \( k \mod (N - 1) \). This function ensures an uniform redistribution on the spare ring composed of the \( N - 1 \) remaining fault-free nodes. The proof that we cannot obtain the faulty node can be found in [7]. The update of managership is ensured at rollback.

### 4.2 Memory Reconfiguration

The memory reconfiguration consists in duplicating lost copies which were located on the faulty node, in order to tolerate again a failure after the rollback. For that purpose, each node scans its memory and for every SH-CK page, checks the node-CK field, which is a field of the page descriptor to know if the corresponding recovery page is still alive or not. If the corresponding recovery copy has been lost, a new one is created on a fault-free node. If the lost copy was a SH-CK1 page, the manager has to be updated with the identity of the new owner. Figure 4 shows the reconfiguration algorithm.

### 4.3 Process Recovery

Besides shared data, it is also necessary to restore the private process states. Each node is responsible for restarting its own processes from the previous recovery point by restoring their recovery context. Moreover, as the permanent failure leads to the unavailability of a node, it is necessary to restart processes which were running on the faulty node. The node which store the second copy of a process recovery point is in charge of its rollback. Then, each node restarts its own processes and those, coming from the faulty node, for which its owns the remote private recovery point.
begin // Memory reconfiguration
For every page p in memory
{case p.state in {
    Shared-CK1: // The node is the owner of the page
    If (node\_CK = faulty\_node) {
        node = (node\_CK + 1) mod N;
        replication(p, node);
        node\_CK = node; // A copy is created on the neighbor node of the faulty one
    }endif
    If (manager\_p = faulty\_node)
        manager\_p = SM(p);
        Update(manager\_p, p, N); // The new manager is updated by the owner of a page p
    }endif
    Shared-CK2: // The node is not the owner of the page p
    If (node\_CK = faulty\_node)
        node = (node\_CK + 1) mod N;
        replication(p, node);
        node\_CK = node;
        Update(manager\_p, p, node); // The manager is updated by the new owner of a page p
    }endif
    default: {skip}
}endcase
}endfor

Figure 4: Memory reconfiguration algorithm

5 Implementation and Results

5.1 Implementation Issues

ICARE has been implemented on a NOW made up of 4 Pentium connected through
an ATM local area network, with each node running the CHORUS [13] microkernel.
The relatively small configuration size is due to the fact that we had only four
workstations equipped with an ATM interface board.

In Chorus terminology, the unit of system resource allocation is called an actor.
It is composed of a protected address space, threads and ports. Threads are the
execution units and are characterized by an execution context (processor registers and
stack pointer). Ports provide a mean of communication between threads, wherever
they are located. There exits two kinds of actor: user actors and supervisor actors.
ICARE is implemented as two CHORUS actors, on each node (see Figure 5). A
user actor is composed of five threads: a user thread which runs the application; a mapper thread which handles page faults generated by the CHORUS microkernel; a communication thread which is responsible for managing communication with the other nodes; a fault-tolerance thread which periodically establishes a recovery point and a rollback thread which is bound to roll processes back, including remote processes, in the event of a permanent failure.

![Figure 5: Overview of ICARE](image)

A supervisor actor, called the context server, is also necessary as it is the only way to use the threadContext() CHORUS primitive which gets and sets the context of a thread. This server may be requested by the fault tolerance thread in order to save a process recovery point and by the rollback thread to restore it.

### 5.2 Performance Results

Experiments have been run on a 4-node configuration with three parallel applications. This choice ensures a wide range of behavior from the viewpoint of memory
access. Matmul is a matrix multiplication algorithm. We have used two working set sizes (256*256 and 512*512) with double float elements for each matrix. The Modified Gram-Schmidt (MGS) algorithm produces from a set of vectors an orthonormal basis of the space generated by these vectors. We consider a base of 512 vectors of 512 double float elements. Radix is a SPLASH-2 sort application. Table 2 depicts the memory operations generated by the considered applications during standard execution, without any fault-tolerance.

<table>
<thead>
<tr>
<th>Application</th>
<th>Number of nodes</th>
<th>Read Faults</th>
<th>Write Faults</th>
<th>Invalidations</th>
<th>Access Rights Growths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matmul 256</td>
<td>4</td>
<td>480</td>
<td>384</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Matmul 512</td>
<td>4</td>
<td>1920</td>
<td>1536</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MGS</td>
<td>2</td>
<td>512</td>
<td>512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1536</td>
<td>512</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RADIX</td>
<td>2</td>
<td>18</td>
<td>15176</td>
<td>15140</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>43</td>
<td>18859</td>
<td>18827</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Memory operations of Matmul, MGS and Radix

To identify the time overhead, we compare results obtained by the basic DSM system and ICARE. The execution time can be divided into the execution time without fault-tolerance mechanism, the time to synchronize all the nodes, the time to replicate pages and change the state of all pages and finally the time which refers to the memory overhead i.e. change of memory behavior due to the introduction in memory of recovery data.

5.2.1 Basic Checkpointing Algorithm

The establishment of a recovery point is likely to involve three particular cases: read on SH-CK2 pages, write on SH-CK2 pages and write on SH-CK1 pages. The former two refer to the use for computation of pages replicated for the fault-tolerance needs whereas the third is overhead due to the establishment of a checkpoint on failure-free executions.

Reads on SH-CK2 As a processor can read a SH-CK2 page, a page may be avoided. It has been anticipated at checkpointing time. Indeed, creating a SH-CK2 copy during a global checkpoint is less expensive than solving a read page fault since many replications are done simultaneously at that time. Furthermore, the systematic way of replication at checkpointing avoids messages exchanged between manager, owner and so on. Tables 3 and 4 and Figures 6 and 7 show that
the occurrence of such read operations may have a significant impact on performance. In Figure 6, the execution time is better with ICARE than with a standard DSM. This phenomenon can be explained by the avoidance of 32% of read page faults. In Figure 7, the fault-tolerance overhead does not exceed 12%. The negative memory overhead due to reads on SH-CK2 pages compensates synchronization and replication times.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>RP/3s</th>
<th>RP/5s</th>
<th>RP/10</th>
<th>RP/15s</th>
<th>RP/20s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read on SHARED-CK2</td>
<td>120</td>
<td>146</td>
<td>136</td>
<td>136</td>
<td>135</td>
</tr>
</tbody>
</table>

Table 3: Memory Overhead for Matmul 256*256 on four nodes

<table>
<thead>
<tr>
<th>Frequency</th>
<th>RP/20s</th>
<th>RP/30s</th>
<th>RP/40s</th>
<th>RP/50s</th>
<th>RP/60s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read on SHARED-CK2</td>
<td>640</td>
<td>564</td>
<td>330</td>
<td>288</td>
<td>242</td>
</tr>
</tbody>
</table>

Table 4: Memory Overhead for Matmul 512*512 on four nodes

![Figure 6. Overhead for Matmul 256*256 on 4 nodes](image1)

![Figure 7. Overhead for Matmul 512*512 on 4 nodes](image2)

**Writes on SH-CK2** Here a write page fault is transformed in a simple access rights growth on readable pages. Access rights growths are less expensive than

Irisa
faults since they do not require page transfers on the network. Figures 8 and 9 and table 5 show that Radix can benefit from such a feature. For the 2-nodes configuration, there is a performance gain of more than 35% compared with a standard execution. This phenomenon is due to the high number of write page faults that are transformed in access rights growths thus anticipating the false sharing of Radix. This latter observation explains the gain increases with a high checkpoint frequency. In the 4-nodes configuration which cannot benefit from the broadcast feature of the 2-nodes one, the overhead is, however, very low. This can be explained by the small number of pages modified between two checkpoints.

![Figure 8. Overhead for RADIX on 2 nodes](image1.png)

![Figure 9. Overhead for RADIX on 4 nodes](image2.png)

<table>
<thead>
<tr>
<th>Frequency</th>
<th>RP/60s</th>
<th>RP/90s</th>
<th>RP/110</th>
<th>RP/120s</th>
<th>RP/150s</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Two Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Miss</td>
<td>9661</td>
<td>11389</td>
<td>11225</td>
<td>12005</td>
<td>14148</td>
</tr>
<tr>
<td>Write on SHARED-CK2</td>
<td>5515</td>
<td>3787</td>
<td>3987</td>
<td>3174</td>
<td>998</td>
</tr>
<tr>
<td><strong>Four Nodes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Miss</td>
<td>19015</td>
<td>18817</td>
<td>18922</td>
<td>19083</td>
<td>18695</td>
</tr>
<tr>
<td>Write on SHARED-CK2</td>
<td>25</td>
<td>20</td>
<td>14</td>
<td>24</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 5: Memory Overhead of RADIX

**Writes on SH-CK1** Such an operation refers to additional access rights growths encountered by the fault tolerance mechanism. Pages which were MOD-
EXC, and thus writable before the establishment of a recovery point, are changed in SH-CK1 and become read-only copies. If such pages are referenced again, this situation involves additional access rights growths. Figures 10 and 11 and Table 6 show that in MGS the large number of write on SH-CK1 explains the high overhead due to the fault tolerance scheme. Every time a page is written before and after a checkpoint, the second write leads to the invalidation of the two SH-CK pages in INV-CK.

![Figure 10. Overhead for MGS on 3 nodes](image)

![Figure 11. Overhead for MGS on 4 nodes](image)

<table>
<thead>
<tr>
<th>Frequency</th>
<th>RP/20s</th>
<th>RP/30s</th>
<th>RP/40s</th>
<th>RP/50s</th>
<th>RP/60s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write on SHARED-CK1</td>
<td>1640</td>
<td>1044</td>
<td>828</td>
<td>708</td>
<td>691</td>
</tr>
</tbody>
</table>

Table 6: Memory Overhead of MGS on 4 nodes

### 5.2.2 Writable Page Optimization Checkpointing Algorithm

The goal of this optimization is to avoid the occurrence of write operations on SH-CK1 pages. Figure 12 shows that this algorithm decreases the huge overhead observed previously for MGS, which now becomes reasonable (around 25%). However,
as shown in Figure 13, this algorithm is not suited for Radix, since SH-CK2 pages issued from writable pages can be exploited in failure-free executions.

![Figure 12. Impact of WPO for MGS](image1)

![Figure 13. Impact of WPO for RADIUS](image2)

### 5.2.3 History-based Checkpointing Algorithm

This algorithm aims at increasing the number of accesses on SH-CK2 pages by creating memory affinity at the establishment of a recovery point. Figures 14 and 15 show that it may have a significant impact on Radix and MGS by increasing the number of write on SH-CK2 pages. Such operations transform write page faults in access rights growths thus improving the efficiency of the DSM system. For Radix, we observe several cases that have never been noticed with the basic algorithm on four nodes, where the execution time with ICARE is better than on a standard DSM. This is due to the anticipation of false sharing (and thus ping-pong effects) by the periodic replication at checkpointing time of data, which are then used for computation.
Figure 14. Impact of Memory Affinity for MGS

Figure 15. Impact of Memory Affinity for Radix

6 Related Work and Conclusions

Many recoverable DSM have been proposed but most rely on the storage of checkpoints on disks and suffer from high latency [6, 18]. The recoverable DSM proposed in [17] also implements stable storage in volatile memories, but recovery data cannot be accessed by processors after a checkpoint. A few systems are related to the same basic principle as ours that is, the combination of fault-tolerance features with standard ones to increase performance. The recoverable DSM presented in [11] takes also benefit of the DSM mechanisms and especially of the entry coherence protocol they use. However, since their approach relies on message logging, they make the assumption of deterministic execution which is too strong in the context of parallel applications. Moreover, this approach does not allow the use for computation of recovery data, which is an important feature of ICARE. This algorithm has also be applied to TreadMarks by exploiting the lazy release consistency protocol [4]. The approach presented in [15] deals with the scheduling of processes in a distributed system during failure-free execution and after a failure. Goals are the assignment and reassignment of processes to processors for which they have a large preference while balancing the load in the system. Nevertheless, affinity is measured instead of being created. In [14], the problem of recovery is considered in a large-scale
transaction-based distributed system. However, this approach is different from ours: data replication introduced in standard behavior to reduce the cost associated with accessing remote data, is used to improve reconfigurations. In ICARE, we exploit data replication induced by fault-tolerance to improve failure-free execution.

The recoverable DSM presented in this paper allows to combine both efficiency and high availability. It demonstrates that fault tolerance and efficiency are not antagonistic in a DSM system. To reach this interesting result, ICARE takes benefits of the characteristics of a basic DSM. Stable storage can be implemented in volatile memories on a NOW. The recovery point establishment algorithm is tightly integrated with the DSM mechanism by simply extending the coherence protocol to manage recovery data and active ones. Moreover, the replication inherent in a DSM system is exploited to limit the cost of fault tolerance. Conversely, ICARE takes benefit of the replication needed in a BER scheme in order to increase efficiency of the DSM system by anticipating page faults. Experimental results show that the proposed algorithms are efficient and we have observed several situations where the execution time with ICARE is lower than with a standard DSM. ICARE also implements the complete rollback recovery in case of a permanent failure and we have presented, apart the memory reconfiguration algorithm which prevents the loss of recovery data and owners, a spare manager function that ensures a uniform distribution of the manager load on the remaining fault-free nodes. Unlike most of proposed recoverable DSM machines, a prototype of ICARE has been completely implemented and has been extensively tested in the presence of failures.

Future work consists in porting our system to Chorus release 3.6 which does not suffer from the drawbacks of release 3.5 in order to test our approach on bigger applications.

Acknowledgments

The author would like to thank Christine Morin for helpfull discussions and comments on her work. The author is also grateful for the carrefull readings of Maarten van Steen and Andy Tanenbaum.

References


