A two-level checkpoint algorithm in a highly-available parallel single level store system

Christine MORIN Renaud LOTTIAUX
Irisa/Université de Rennes 1, Rennes, France
{cmorin,rlottiaux}@irisa.fr

Anne-Marie KERMARREC
Microsoft Research, Cambridge, UK
annemk@microsoft.com

Abstract

A Parallel Single Level Store systems (PSLS) integrates a shared virtual memory and a parallel file system. Managing globally the data, they provide programmers of scientific applications with the attractive shared memory programming model combined with a large and efficient file system in a cluster. In this paper, we present a cheap and efficient two-level checkpointing approach enabling a PSLS to tolerate failures. The first level checkpointing algorithm is very efficient and saves data in memory but requires a large amount of memory space. When memories are saturated, an alternative algorithm, saving a checkpoint on disks is implemented. Performance results present the impact of different variants of the checkpointing algorithms.

1 Introduction

A Parallel Single Level Store system (PSLS) is particularly attractive for the execution of long-running parallel applications, such as numerical simulations on clusters of workstations. Effectively, such applications are often based on the natural shared memory programming model and perform a large amount of inputs and outputs. To cope with this twofold requirement, a PSLS integrates a Shared Virtual Memory (SVM) [1, 11], thus providing programmers with a simple and attractive programming model, and a Parallel File System (PFS) [7, 15] to enable a large and efficient disk storage. PFSs usually provide high disk bandwith by fragmenting a file on several disks of different cluster nodes, enabling parallel accesses to fragments. Unfortunately, interfaces are quite complex which is contradictory with the presence of a simple shared memory programming support. A PSLS provides a simple mapping interface to a PFS: parallel files are mapped into a global virtual address space and all file operations are implicitly performed through memory reads and writes in the SVM thus releasing programmers from explicitly managing data transfers between disks and memories. Moreover, concurrent accesses to the same data file are automatically managed by the SVM coherence protocol [13]. In the global address space, the volatile data, which life time is the duration of the computation, cohabit with mapped data, which have a counterpart in the PFS. This paper deals with the interactions between an SVM and a PFS. Thus, we focus on the mapped data management.

PSLS systems are implemented on clusters that are made up of a large number of components and thus are vulnerable to failures which is unappropriate for long-running applications. To deal with this, we propose a PSLS providing an efficient support for high-availability. Our system does not require any specific and expensive hardware and relies on a Backward Error Recovery (BER) approach where a coherent state of the system is periodically snapshot and stored in stable storage and restored in the event of a failure.

In this paper, we present an efficient two-level checkpointing approach where checkpoints are established as much as possible in memory thus avoiding costly disks operations according to the first level algorithm. However this algorithm requires a large amount of extra memory for the duration of the checkpoint creation and the system may reach a deadlock situation if memories become saturated. We propose an alternative algorithm to solve this potential bottleneck problem (second level algorithm). Moreover the implementation of this algorithm, called the *cleaner* checkpoint algorithm provides additionnal guarantees in term of fault-tolerance. Indeed, it enables to tolerate power cut failures in the cluster. We study the impact on performance of several variants of the cleaner checkpoint algorithm. Note that the whole checkpointing algorithm deals with volatile and mapped data of the PSLS as well as with the private state of each application process. The checkpointing approach for volatile data as well as process checkpoints were previously investigated in the framework of ICARE recoverable SVM [8]. In this paper, we only focus on mapped data management. The remainder of the paper is organized as follows: Section 2 presents our system model and fault-tolerance assumptions, Section 3 depicts the checkpointing alorithms, the evaluation of the variants are presented in Section 4. Section 5 describes related works before the con-

0-7695-1010-8/01 $10.00 © 2001 IEEE

clusion.

2 System Model

Data management A PSLS defines a global virtual address space on top of distributed nodes, giving the illusion of a global shared memory (including memory and PFS contents). The unit of access and transfer on this global address space is the memory page.

An overview of a PSLS system is depicted on Figure 1. Each node is composed of a processor, a memory and a PFS disk. In this example, node 0 stores pages 0, 2 and 4, whereas node 1 stores pages numbered 1 and 3 in their PFS disk. However, PFS pages mapped into the PSLS global address space at initialisation are accessible to all nodes. Nevertheless, they are not loaded in memory. Afterwards, when a page is referenced, through a standard read or write operation, if the page is already loaded, standard SVM mechanisms are used to retrieved or replicate the page, otherwise, the PSLS management downloads the page from the PFS.

Data replication in an SVM leads to the presence of several copies of a page in different memories. A coherence protocol managing both mapped and volatile pages is implemented in order to ensure the consistency of multiple copies. Our PSLS is based on sequential consistency model [4] implemented with a write-invalidate protocol.

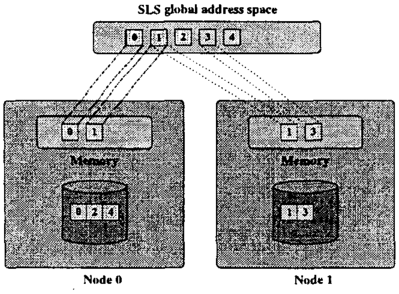

Figure 1. PSLS architecture

High-availability assumptions We consider a system composed of failure-independent nodes operating in a fail-stop mode [16]. Our system is able to tolerate simultaneous transient failures and a single permanent fault leading to the loss of one node contents (memory and PFS disk) as well as a computing node. The failure of a node component leads to the unavailability of the whole node. Nodes are failure-independent. We assume that nodes are connected by a reliable interconnection network. Our system relies on backward error recovery [10] which is a recovery technique well-suited to high performance parallel applications: a consistent system state (a checkpoint) is periodically saved and stored on stable storage and restored upon detection of a failure. A stable storage ensures that (1) data is not altered and remains accessible despite a failure (*persistence* property), and that (2) data is updated atomically in presence of failures (*atomicity* property).

The coherence of the checkpoint is ensured by an incremental global coordinated checkpointing policy where all nodes save simultaneously a checkpoint [2]. A two-phase commit protocol [5], where a previous checkpoint is invalidated only when the new one is validated, guarantees the atomic update of a checkpoint.

3 A two-level checkpointing approach

The ultimate goal of our system is to provide an efficient checkpointing algorithm. No dedicated hardware is required, stable storage is simply ensured by replicating every page on two different nodes. We designed a two-level checkpointing approach: an efficient algorithm, the **memory checkpoint** is used by default and creates a checkpoint in memories. Unfortunately, establishing a memory checkpoint is not always possible due to lack of memory space and an alternative algorithm is proposed: **the cleaner checkpoint** algorithm. We detail these two checkpoint algorithms in the following.

3.1 Memory checkpoint

In order to efficiently establish a checkpoint, a default and efficient checkpoint algorithm called memory checkpoint is proposed which exploits the high bandwidth between memories in a cluster. In the PSLS, each page has two copies called **recovery copies** belonging to the checkpoint. Establishing a new checkpoint consists in creating two new recovery copies for each page that has been modified since the last checkpoint. A page is **clean** if its copies in memory are identical to the disk copy. If the disk copy of a page is not up-to-date (the up-to-date copy being in memory), this page is called a **dirty** page. Pages that have to be checkpointed are thus those that have been loaded from the PFS disks into memory and that have been modified since the last checkpoint.

The stability of recovery copies stored in memory is implemented by having two recovery copies for each page that are located in two different node memories. The stability of data stored on the PFS is implemented with a mirroring mechanism.

No disk access is needed to establish a memory checkpoint as new recovery data is created in memory. The memory checkpoint algorithm is thus efficient.

Data stored in memory belongs to one of the following category:

- **Active data** is data used for computation and which does not belong to a checkpoint.

- **Readable recovery data** represents recovery data not modified since the last checkpoint. It remains readable and can be used for standard execution as long as it is not modified. Upon modification it becomes pure recovery data.

- **Pure recovery data** represents recovery data, no longer usable for standard execution, this data is restored in the event of failure.

A memory checkpoint is composed of recovery pages present in memory (pure or readable) and pages stored on the PFS which have no recovery copy in memory. This decomposition ensures the coherence of a checkpoint.

The memory checkpoint algorithm is very similar to the one presented in details in [8] for a SVM system. In the SVM, two kinds of page exist: unique and replicated pages. A page is unique if it is writable or, readable but not yet replicated. During the first phase of the two-phase commit algorithm, new recovery data is created. During this phase, performed in parallel by each node, the recovery data belonging to the previous checkpoint and the recovery data belonging to the on-going checkpoint cohabit in memory. The algorithm works as follows:

- **Unique pages**: each unique page is transformed into a *pre-recovery data*, which represents data belonging to the on-going checkpoint but not yet validated. Moreover, a pre-recovery replica of the page is created on another node.

- **Replicated pages**: for each page present in several memories, two replicas are transformed into pre-recovery copies. This optimization saves some memory space and avoids data transfers on the network as well, thus increasing the overall efficiency of the memory checkpointing algorithm.

The second phase is local to each node. Pure recovery pages belonging only to the previous checkpoint are discarded. Readable recovery data belonging to both checkpoints remains unchanged and pre-recovery data is transformed into readable recovery data.

If a failure is detected before the end of the first phase the previous checkpoint (composed of pure and readable recovery data) is restored. If the failure is detected during the second phase, the new checkpoint (composed of readable recovery and pre-recovery data) is restored.

During the first phase, potentially four memory items are required to ensure that one page is correctly checkpointed: for instance, two pure recovery data belonging to the previous checkpoint and two pre-recovery data belonging to the new checkpoint may exist in the system. This constraint is quite strong and the system may reach a deadlock situation where not enough room is available to establish a checkpoint. To cope with this situation, we propose an alternative algorithm which involves disk operations and enables to empty the memory while taking simultaneously a checkpoint: this algorithm, called the cleaner algorithm writes back some data into the PFS, establishes a persistent checkpoint and ensures that memory checkpoints are going to be possible in the future.

3.2 Cleaner checkpoint algorithm

PFS disks implement mirroring. At initialization, when a page is created in the PFS, two copies are created, a primary and a mirror copy, on two distinct nodes.

The goal of the cleaner algorithm is to clean the memory by writing back into PFS mapped pages that have been loaded and modified into memory at their PFS address (on place). Since in this algorithm disk accesses are unavoidable, we take benefit of this algorithm to save a checkpoint exclusively on disks. This way, we implement a persistent checkpoint which enables to tolerate power cut failures affecting the whole cluster. As a consequence, PFS disk write operations are not enabled between checkpoints and are only performed during the cleaner checkpoint algorithm.

The algorithm works as follows: all dirty pages are copied back on their correponding PFS disk which persistence is ensured by the use of mirroring in the PFS. Clean active mapped pages do not need to be checkpointed. Dirty mapped pages need to be written back on the PFS and the PFS mirror. Readable recovery pages need to be written back on the PFS as well, in order to guarantee the consistency of the checkpoint. Finally, pure recovery pages are simply discarded since the corresponding modified active page is checkpointed.

The cleaner algorithm is also performed according to a two-phase commit algorithm thus ensuring the atomicity of the checkpoint and the atomic write operation of the primary and mirror PFS pages. The cleaner algorithm fullfills two functions: cleaning the memory and establishing a persistent checkpoint. The cleaner algorithm could be launched on demand when a deadlock situation is about to occur (such a situation can be detected by maintaining counters on the memory occupation centralized on the coordinator of checkpointing for example). It is also called periodically to establish a persistent checkpoint.

Example

In this section, we present an example illustrating the functioning of the PSLS. Figure 2 represents the considered initial state of the PSLS. There are 5 mapped pages. Page 0, 1 and 2 are loaded in node 0 and node 1 memories. Page 3 is loaded only in the memory of node 1 and page 4 is loaded in the memory of node 0. Plain, striped and squared squares with the same number represent different versions of the same page. White squares represent recovery data and grey squares active data. On disks, only copies represented by white squares belong to the current memory checkpoint. All disk copies belong to the last persistent checkpoint. Readable recovery copies are located in the first line in memory while pure recovery copies are located in the second line in memory. Initially, page 0 corresponds to a page that has been loaded in memory, modified, checkpointed in memory and modified afterwards. It is shared by nodes 0 and 1 and thus two dirty active copies exist in memory. Page 0 has two pure recovery copies in memory and its disk copies are not up to date. Page 4 is similar to page 0 except that it is not currently shared thus a single copy exist in memory. Page 1 has been modified, checkpointed in memory and not modified after the last checkpoint. Thus, there are two readable recovery copies corresponding to page 1. Disk copies are not up to date. Page 2 has been modified after the last checkpoint. Thus it is not yet been checkpointed in memory. Recovery copies of dirty page 2 are its two disk copies. Page 3 has been loaded in the memory of node 1. It has not been modified thus the contents. For clean page 3, the recovery copies are the disk copies. Their contents is identical to the contents of the memory copy.

Each page is located in two disks as mirroring is implemented.

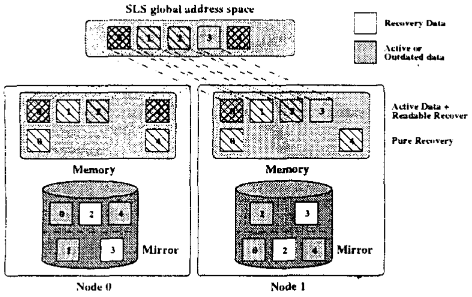

Figure 2. *Initial state*

Figure 3 represents the state of the PSLS after a memory checkpoint has been saved. New recovery copies have been created only for modified active pages, namely page 0, 2 and 4. The two existing copies of page 0 and 2 have been exploited and simply transformed into readable recovery copies. As only one copy exists in memory for page 4, a new copy is created in node 1. Existing readable copies of page 1 are left unchanged. Pure recovery copies are discarded as they do not belong to the new checkpoint. The disk is not accessed during a memory checkpoint. However, as page 2 now has recovery copies in memory, its disk copies do not belong anymore to the current memory checkpoint and are now grey. Figure 4 represents the state of the PSLS after a persistent checkpoint has been saved (with no cleaning). All dirty pages are copied to disk to ensure the existence of a coherent persistent checkpoint on disks. Dirty pages 0, 2, 4 are copied back to disk. Page 2 readable recovery copies need also be copied back to disk as they belong to the new checkpoint. No action is required for page 3 which is clean.

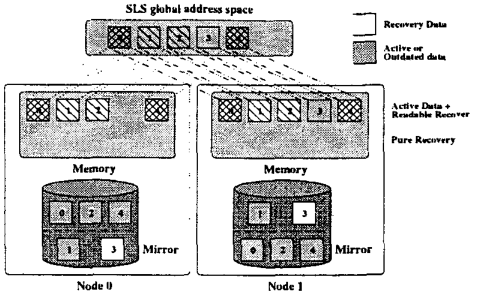

Figure 3. *New state after a memory checkpoint*

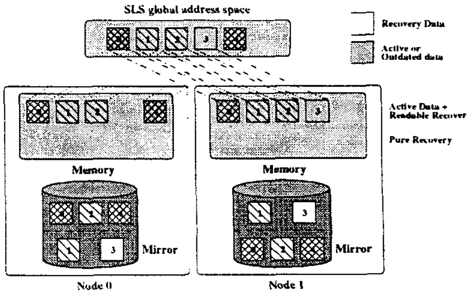

Figure 4. *New state after a persistent checkpoint*

Cleaner strategies

Several variants of the cleaner algorithm may be considered. However, for each of these variants, a persistent checkpoint is taken onto disk. Basically these variants differ by the state of the memories after a checkpoint has been established as regard to what extent it reflects the working set of application processes.

The most radical form is to empty completely the memory: this way, an application looses its working set that has to be loaded again from the PFS which remains quite costly.

The other extreme would be to keep all pages in memory, after having discarded pure recovery data that are not relevant anymore. This way, the application working set is kept thus reflecting the application access patterns. However, the memory is alleviated only from the pure recovery data, which may be not enough to be able to establish a memory checkpoint later on.

Several intermediate variants can be defined, such as keeping all writable pages, or all writable pages plus one copy of a readable page, or removing pages until a treshhold is reached according to a LRU standard algorithm.

In this paper we study five variants. We consider two extreme policies : the no cleaning policy in which no active and readable recovery copy is discarded from memory and the total cleaning policy in which all active and readable recovery copies are evicted from memory. Three additional variants have been studied. The first cleaning policy keeps in memory only one copy for each clean page, two copies for each dirty active mapped page which are two readable recovery copies and the two readable recovery copies for each page previously checkpointed in memory. The second cleaning policy keeps only one copy for each clean page, two readable recovery copies for each dirty mapped active page. It discards readable recovery copies belonging to the previous checkpoint. The third cleaning policy keeps only one active copy for each page loaded in memory and keeps readable recovery copies of the previous checkpoints.

3.3 Rollback recovery

Memory and persistent checkpoints are established periodically, the period for persistent checkpoints obviously being much larger. In addition, when memories are saturated and a memory checkpoint is no longer contemplated, a cleaner checkpoint algorithm is called, even if the period has not elapsed between two persistent checkpoints.

When a transient or a persistent failure is detected, the system is rolled back to the last checkpoint whether it is a memory or a persistent checkpoint and the treatment slightly differs. When a power cut occurs, the system is rolled back to the last persistent checkpoint.

Handling transient failures

Upon detection of a transient failure, if the last checkpoint is a memory checkpoint, all dirty active pages present in memory are discarded, pure recovery pages are restored into readable recovery data whereas readable recovery data as well as clean copies remain unchanged.

If the last checkpoint is a persistent one, memory pages (recovery or dirty active data) are discarded, since the recovery data is the data stored on disk. Note that clean pages need not be evicted from memory.

Handling permanent faults

Upon detection of a permanent fault, the same rollback procedure is applied as in the case of a transient failure. Nevertheless, in addition, the faulty node has been lost as well as its disk and memory contents. The memories should then be reconfigurated in order to be able to tolerate again a failure righ away. To this end, each page, whose recovery replica was stored on the faulty node memory or respectively disk, needs to be replicated on another memory node, respectively disk node. Moreover a spare manager (SVM and PFS) must be defined for pages which were managed by the faulty node. For a complete description of the rollback, see [8].

In the event of a persistent checkpoint restoration for both transient and permanent faults, all dirty pages are evicted from memory. This means that after the rollback, disk accesses may be required in order to rebuild the modified part of the working set of the application. To avoid this, it is possible to anticipate and to load in advance the pages which were active, from the PFS. The same strategies as the ones driving the cleaner algorithm variants may be considered to select the data to prefetch.

When a power cut failure occurs, a persistent checkpoint is restored, the memory contents being lost. In such a case, application working sets need to be completely reloaded from disks.

4 Evaluation of cleaner algorithm variants

4.1 Prototype overview

We have implemented a user-level prototype of our highly-available PSLS. The prototype has been developped on a cluster of dual processors PCs. The prototype consists in an integration of a sequential consistency-based SVM and a basic PFS. The coherence and access management unit in the SVM and the PFS is equal to the size of a memory page (4KB). The current prototype does not implement the checkpointing of the application processes kernel-internal state. Nodes are based on Pentium II (450 MHz) and has a 256 M-byte local memory. The system is based on the Scalable Coherent Interface SCI [6]. In the prototype, the SCI network has a latency of about 5 microsecond and a throughput of about 60 M-bytes per second. Performance results have been obtained from the execution of Modified Gram Schmidt (MGS) algorithm on a four node cluster. The MGS algorithm produces from a set of vectors an orthonormal basis of the space generated by these vectors. We consider a base of 1024 vectors of 1024 double floats elements. MGS vectors are stored in a PFS file mapped in the SVM.

4.2 Memory versus cleaner checkpointing algorithm

Figure 5 shows a comparison of the execution times obtained with MGS with the memory checkpoint algorithm and

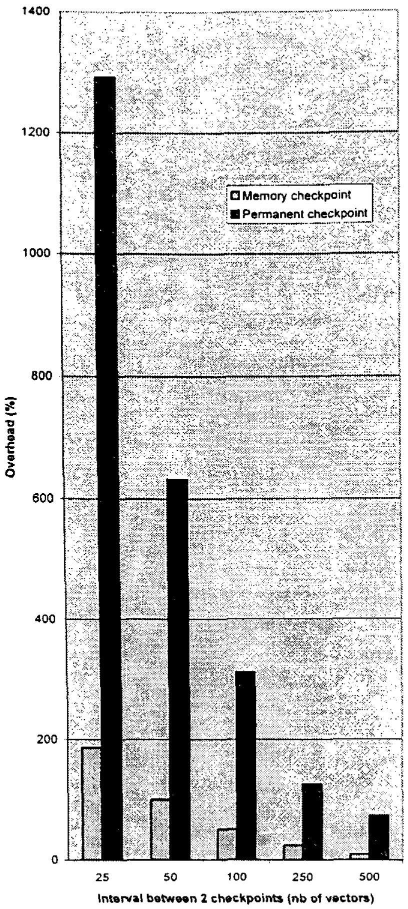

Figure 5. *Comparison between memory and disk checkpoints*

with the persistent checkpoint algorithm. Different checkpoint frequencies have been used for the experimentations. The call to the checkpoint primitive is placed at the beginning of the main loop of MGS. So, the frequency is given as an interval between checkpoints expressed in number of vectors. A frequency of 250 means that a checkpoint is saved each time 250 vectors have been computed (it corresponds to a checkpoint every 4 seconds). An overhead of 100% means that the execution time of the fault tolerant version of the application is twice the one of the standard version.

We see that the overhead increases with the number of checkpoints. Indeed, each time a checkpoint is saved a global synchronization of all processors is needed. Moreover, a greater amount of recovery data is dealt with in MGS when the checkpointing frequency increases. Checkpointing in memory is very efficient as new recovery data need only to be saved for pages modified since the last checkpoint and still in memory. The checkpointing time increases considerably when a persistent checkpoint is saved as a lot of disk accesses are performed. For instance, at a frequency of 100 for the *persistent* checkpoint algorithm, 512 pages are copied by each node into the PFS. However, a persistent checkpoint is not intended to be often saved. The interval between two persistent checkpoints should be at least some minutes in a realistic system.

4.3 Variants of the cleaner checkpointing algorithm

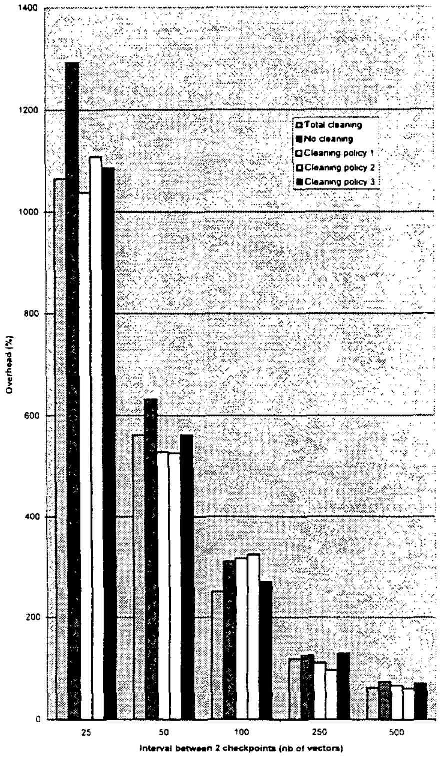

Figure 6. *Comparison of different variants of the cleaner checkpoint algorithm*

Figure 6 shows the execution time of MGS for different variants of the cleaner checkpoint algorithm and for various checkpoint frequencies. We observe that the memory behavior is altered in several policies. This has an impact for high frequencies essentially. This is due to the fact that modified active copies that were writable before a checkpoint are transformed into readable recovery copies (read-only) during the checkpoint. This generates a large number of access right growths.

5 Related Work

Very few other works has been done on file mapping in software DSM with a PFS [9, 12] but none of them is designed to tolerate node failures.

Several recoverable software shared virtual memory systems have been proposed [14]. In [3], a reliable remote paging mechanism is described for a network of workstations. However, to our knowlegde, all these memory management systems do not consider issues related to the interactions between the memory management system and a (parallel) file system.

XFS [17] is a highly available parallel file system which implements cooperative caching. In contrast to our system, memory and disk management is not integrated. Moreover, XFS provides a standard read/write interface and implements RAID-5 rather than mirroring to ensure the high availability of files. Thus complex mechanisms (management of write logs) are needed to efficiently implement a distributed RAID-5 mechanism.

6 Conclusion

In this paper, we have proposed an efficient two-level checkpointing mechanism for a PSLS integrating a SVM and a PFS. This approach enables a cluster to tolerate transient, permanent as well as power cut faults without reqiuiring any specific harware. The default checkpoint algorithm establishes efficiently a checkpoint in memory, its main limitation is the memory space it requires to be implemented correctly. To overcome this drawback, we propose an alternative checkpointing algorithm, called *cleaner*, based on disk ressource usage. The cleaner algorithm enables (i) to *clean* the memories when they are saturated in order to be able to establish memory checkpoints later and, *(ii)* to establish a persistent checkpoint on disk. This enables to tolerate power cut failure of the whole cluster.

We have implemented these algorithms in a preliminary prototype and have studied the impact on performance of these two algorithms: the memory algorithms and the different variants of the cleaner algorithm. Not suprinsingly, experience shows that checkpointing the data in memory is much more efficient than on disks. Thus, the frequency of disk chekpoint as well as the cleaning policy must be chosen carefully to keep the cost reasonable. Future work includes further study to select the most appropriate cleaning policy depending on the access patterns of the application.

References

[1] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. TreadMarks: Shared Memory Computing on Networks of Workstations. *Computer*, pages 18–28, February 1996.

[2] K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3 (1):63–75, February 1985.

[3] E. Marcatos G. Dramitinos. Adaptative and reliable paging to remote main memory. *Journal of Parallel and Distributed Computing*, 1999. to appear.

[4] M. Dubois, C. Scheurich, and F.A. Briggs. Synchonization, coherence and event ordering in mu ltiprocessors. *IEEE Computer Survey, Tutorial Series*, February 1988.

[5] J. Gray. *Notes on Database Operating Systems.*, volume 60 of *Lecture Notes in Computer Science*. Springer Verlag, 1978.

[6] D. B. Gustavson. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro*, pages 10–21, February 1992.

[7] J.V. Huber, C.L. Elford, D.A. Reed, and A.A. Chien. PPFS: A high performance portable parallel file system. In *Conference proceedings of the 1995 International Conference on Supercomputing*, pages 385–394. ACM Press, July 1995.

[8] A.-M. Kermarrec, C. Morin, and M. Banâtre. Design, implementation and evaluation of ICARE: an efficient recoverable dsm. *Software Practice & Experience*, 28(9), July 1998.

[9] O. Krieger, K. Reid, and M. Stumm. Exploiting mapped files for parallel i/o. In *SPDP Workshop on Modeling and Specification of I/O*, October 1995.

[10] P.A. Lee and T. Anderson. Dependable computing and fault-tolerant systems, v ol. 3. In J.C. Laprie A. Avizienis, H. Kopetz, editor, *Fault Tolerance : Principles and Practice*. Springer Verlag, New York, 1990.

[11] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *IEEE Transactions on Computers*, 7(4):321–359, November 1989.

[12] Qun Li, Jie Jing, and Li Xie. BFXM: A parallel file system model based on the mechanism of distributed shared memory. *ACM Operating Systems Review*, 31(4):30–40, October 1997.

[13] C. Morin and R. Lottiaux. Global resource management for high availability and performance in a dsm-based cluster. In *Proc. of 1st workshop on Software Distributed Shared memory*, June 1999.

[14] C. Morin and I. Puaut. A survey of recoverable distributed shared memory systems. *IEEE Transactions on parallel and Distributed Systems*, 8(9), September 1997.

[15] N. Nieuwejaar and D. Kotz. The Galley parallel file system. In *Proceedings of the 10th International Conference on Supercomputing*, pages 374–381, August 1996.

[16] R.D. Schlichting and F.B. Schneider. Fail-stop processors: an approach to designing fau lt-tolerant computing systems. *ACM Transactions on Computing Systems*, 1:pp 222–238, August 1983.

[17] T. Anderson M. Dahlin J. Neefe D. Patterson D. Roselli R. Wang. Serverless network file systems. In *proc. of 15th ACM Symposium on Operating Systems Principles*, December 1995.