cation of online and offline archives. Diskless, single disk, and machines without offline storage are supported by the use of networked resources for logging and archiving.

Stable Transactional Memories and Fault Tolerant Architectures

M. Banâtre Ph. Joubert Ch. Morin
G. Muller B. Rochat P. Sanchez
e-mail: banatre@irisa.fr

IRISA Campus de Beaulieu,
35042 Rennes cedex (France)

1 Introduction

In order to provide powerful and reliable computers, many fault tolerant multiprocessor architectures have been developped in recent years. These architectures can be subdivided into two categories :

- loosely coupled category in which each processor has private ressources and communicates with other processors through message passing protocols,
- tightly coupled category in which each processor directly accesses all the memory and I/O ressources and communicate with other processors through shared memory.

Apart from N-modular redundancy techniques like the one proposed in System/88 [Harr87] or FTMP [Hopk78] computers which are valid for all types of multiprocessors, different means to tolerate processor failures are adopted dependent upon the type of architecture.

One method of achieving fault tolerance on loosely coupled architectures uses a process-pair scheme. Every process running in the system is backed up on another processor. A primary process executes the main computation and is periodically synchronized with an inactive backup process that holds checkpoints. A checkpoint can be defined as a process state from which computation may be safely restarted if the processor running the primary process fails. Tandem [Bart87] and the recent Targon/32 [Borg89] systems are representative examples of this approach.

The Sequoia architecture [Bern88] is a tightly coupled fault tolerant multiprocessor not using N-modular redundancy techniques. In this architecture, a non write-through cache memory is associated with each processor. This allows multiple writes on a memory block before the main memory cache is updated. Each processor locally performs memory updates within its cache and periodically checkpoints its state by flushing the cache and its internal registers to main memory. Modified data is flushed on two distinct memory modules to handle memory and processor failures during a flush operation.

At IRISA, we propose a new approach : the design of a fault tolerant architecture based on a specialized memory : the Stable Transactional Memory (STM). The STM contains a set of consistent structures which can only be handled inside transactions.

2 The Stable Transactional Memory

The concept of Stable Transactional Memory is the result of our investigation in the use of the fast stable storage technology to build reliable applications. In our first experience, the implementation of the ENCHERE system [Bana86a], fast stable storage memories have been used to provide efficient commit protocols. A second experiment of this concept has been studied in the GOTHIC system, the purpose of this system is to provide a distributed operating system on a network of loosely coupled multiprocessor architectures.

The main features of this second release of the stable storage can be summarized as follows [Bana88] :

autonomy, the stable storage board is able to take some decisions. For instance, it can decide that a processor, accessing it, is faulty and then initiates a reconfiguration.

auto-protection, the stable storage has mechanisms to protect itself against a processor failure.

atomic transactions, the stable storage board provides an atomic transaction facility (atomic sequences of read or write accesses) on groups of objects.

This stable storage has been integrated into a multiprocessor architecture (fig 1) by associating a stable storage board to each processor. Every process in the system can allocate stable memory and access stable objects (e.g. checkpoints). No backup process is needed since the stable memory board associated with each processor can be accessed by another processor through the global bus to retrieve stable objects (e.g. checkpoint) after a processor crash.

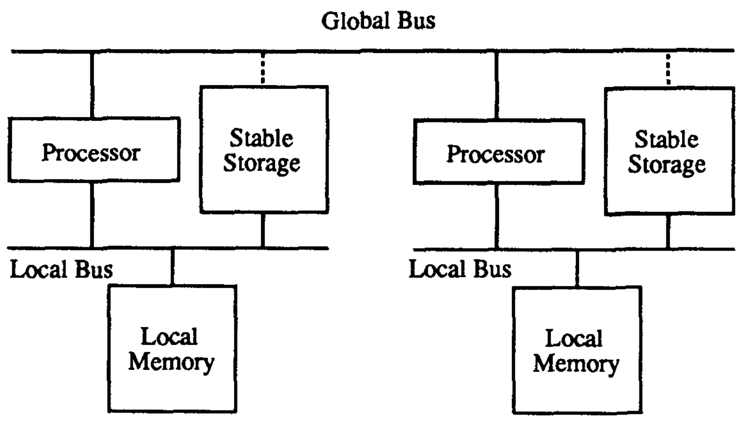

Figure 1: A GOTHIC multiprocessor architecture

Currently, many reliable applications have been built on this architecture, in particular a reliable stable memory manager [Mull88] and a reliable communication subsystem [Bana89], [Mori90] are implemented.

The GOTHIC experience has convinced us that programming reliable applications takes advantage of the built-in atomic transactions features of the stable storage. However, the multiprocessor we used could not be extended to provide a real fault tolerant multiprocessor architecture, (only one bus, only one transaction at a time...), so we decided to investigate more deeply the fault tolerant architectures based on the stable transactional memory concept. This concept can be considered as the generalization of the stable memory provided in the GOTHIC architecture. In the next sections we consider how this concept can be used to construct fault tolerant architectures.

3 Thightly coupled fault tolerant multiprocessor architecture

Traditionnally, thightly coupled multiprocessors allow data sharing between multiple caches by keeping cached copies of memory blocks coherent with respect to shared memory. This is difficult to achieve in a fault tolerant environment due to the need to save global checkpoints in shared memory. Current solutions avoid this problem, they forbid data sharing between caches [Bern88]. Our solution to this problem is based on the use of a stable transactional memory (STM) instead of the common memory (fig 2). The atomic transaction (or atomic action) provided by the STM is very close to those described in [Lamp81], but at the cache level.

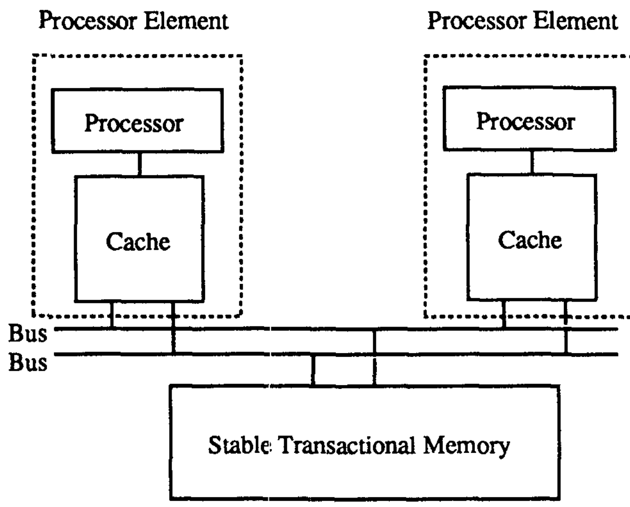

Figure 2: A fault tolerant tightly coupled multiprocessor architecture

In our architecture, process states are locally modified within non write-through caches and atomically updated into the STM using the hardware transaction mechanism provided by the STM itself. Sharing of memory blocks between multiple processes is handled by a cache coherence protocol very close to the one used in standard shared memory multiprocessors. Caches log dependencies between processes sharing memory blocks and dependent process states are updated atomically. Here we do not detail the protocols we have designed, they are described in [Bana90a].

4 Fault tolerance for open systems architecture

The Fault Tolerant Multiprocessor (FTM) [Bana90b] system aims at providing a general purpose open system in which hardware and operating system failures are masked to the user.We identify three main subjects of interest : special hardware support for standard architecture, design of a reliable kernel, fault transparence to user programs.

Special hardware support

Our goal is to design fault tolerant computers from various standard architecture components. This approach is very similar to the GOTHIC one and consists in associating a STM board to every processor (Node). The overall principle is to have redundant access path to each subsystems (STM, mirrored disks, ..). The actual FTM prototype (fig 3) is built from two loosely coupled multiprocessors interconnected by a fault tolerant link (FTL). Every STM board possesses a recovery access only activated by the STM in presence of crash of its processor. Two processors of the two multiprocessors are coupled by their STM recovery access to form a Stable Pair of Nodes (SPN). In normal service, no node is dedicated to backup. On the other hand if one node fails, the other node of the SPN resumes its work.

The reliable kernel

The major aspects of the kernel are protection, fast restart and portability. Before accessing or modifying any stable object, the programmer must call an "open_object" command (provided by

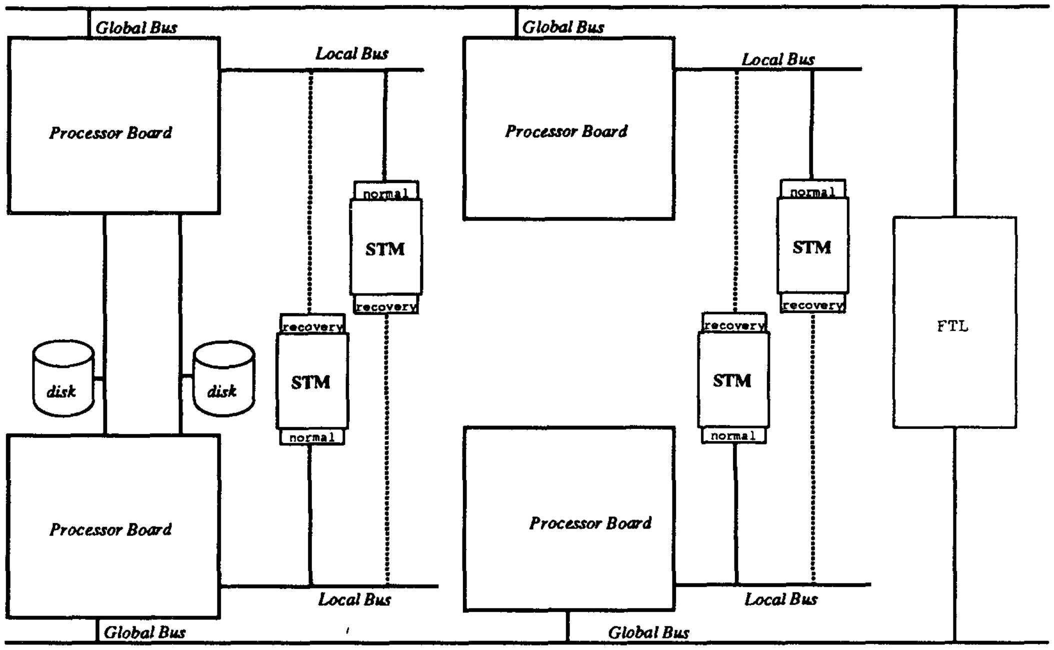

Figure 3: Architecture of the FTM

the STM). Any reference to a close object is detected by the STM which signals its occurence to the kernel. Operations on stable objects have to be part of STM transactions. In case of failure, the transaction is aborted and the object initial states are restored by the STM. Then, the second node of the SPN can restart computations directly from STM objects, without repairing any erronous state.

To enable generalization of the FTM, the kernel has to be easily portable. In order to do so, the FTM kernel is compatible with and built from the MACH/OSF1 kernel [Acce86]. Some standard services of OSF1 as virtual memory and I/O handlers are replaced by reliable ones which are based on STM transactions.

Fault transparency

A user program can be designed in two different ways on the FTM. First, a reliable program can be written using STM objects and transactions. Second, an existing software may be adapted to the FTM. Thus, the hardware faults must be masked to the user. This is acheived by the Distributed Shared Memory sub-system Management which is a secure Single Level Store derived from the GOTHIC one [Mich89] [Roch90].

5 Concluding remarks

This short note has reviewed our ongoing researches on fault tolerant architectures based on the Stable Transactional Memory mechanism. We have just completed the design of the hardware architecture of the FTM and currently we are investigating the integration of STM facilities into operating system kernels in order to make them fault tolerant.

References

[Acce86] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young.

Mach: A New Kernel Foundation for UNIX Development. In *Proc. of Usenix 1986 Summer Conference*, July 1986.

[Bana86a] J.P. Banâtre, M. Banâtre, G. Lapalme, and Fl. Ployette. The Design and Building of ENCHERE, a Distributed Electronic Marketing System. *Communications of the ACM*, 29(1):19–29, January 1986.

[Bana88] J. P. Banâtre, M. Banâtre, and G. Muller. Ensuring data security and integrity with a fast stable storage. In *Proc. of 4th International Conference on Data Engineering*, pages 285–293, IEEE, Los Angeles, February 1988.

[Bana89] J. P. Banâtre, M. Banâtre, and C. Morin. Implementing Atomic Rendezvous within a Transactionnal Framework. In *Proc. of 8th Symposium on Reliable Distributed Systems*, pages 119–128, IEEE, Seattle, October 1989.

[Bana90a] M. Banâtre and P. Joubert. Cache Management in a Tightly Coupled Fault Tolerant Multiprocessor. To appear in *Proc. of 20th International Symposium on Fault-Tolerant Computing Systems*, Newcastle, June 1990.

[Bana90b] M. Banâtre, G. Muller, B. Rochat, and P. Sanchez. *An overview of the FTM system.* Research report, INRIA, Rennes (France), to appear 1990.

[Bart87] J. Bartlett, J. Gray, and B. Horst. *Fault Tolerance in Tandem Computer Systems*, pages 55–76. Volume 1, Springer Verlag, 1987.

[Bern88] Ph. A. Bernstein. Sequoia: A Fault-Tolerant Tightly Coupled Multiprocessor for Transaction Processing. *IEEE Computer*, 37–45, February 1988.

[Borg89] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle. Fault Tolerance under UNIX. *ACM Transactions on Computer Systems*, 7(1):1–24, 1989.

[Harr87] E. S. Harrison and E. Schmitt. The Structure of SYSTEM/88, a Fault-Tolerant Computer. *IBM Systems Journal*, 26(3):293–318, 1987.

[Hopk78] A. Hopkins. FTMP: A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft. *Proc. IEEE*, 66(10):1221–1239, October 1978.

[Lamp81] B. Lampson. Atomic Transactions. In *Distributed Systems and Architecture and Implementation : an advanced course*, Lecture Notes in Computer Science, 1981.

[Mich89] B. Michel. *Conception et réalisation de la mémoire virtuelle de GOTHIC.* Thèse de doctorat, University of Rennes I, September 1989.

[Mori90] C. Morin. *Conception et réalisation d'un service d'appel de multiprocédure à distance.* Thèse de doctorat, University of Rennes I, 1990, in preparation.

[Mull88] G. Muller. *Conception et réalisation d'une machine multiprocesseur sûre de fonctionnement.* Thèse de doctorat, University of Rennes I, June 1988.

[Roch90] B. Rochat. Implementation of a Shared Memory System on a Loosely Coupled Multiprocessor. In *Proc. of 5th Distributed Memory Computing Conference*, IEEE, Charleston, April 1990.