Global and Integrated Processor, Memory and Disk Management in a Cluster of SMP's

Christine Morin
IRISA/INRIA
Campus universitaire de Beaulieu, 35042 Rennes cedex (FRANCE)
(cmorin@irisa.fr)

Abstract

Due to the increasing power of microprocessors and the evolution of networking technology, clusters of SMP's are now attractive for executing data and/or computation intensive applications. A key problem is to ensure both high performance and high availability properties that are needed for proper execution of long-running parallel applications on a cluster. In this paper, we present the design of Gobelins, a cluster operating system, aiming at providing these two properties to parallel applications based on the shared memory programming model. Gobelins relies on global and integrated management of all memory, disk and processor resources in a cluster. It is currently under development as an extension of Linux operating system for a cluster of dual processor PCs interconnected by a low latency high bandwidth network.

1 Introduction

Due to the increasing power of microprocessors and the evolution of network technology, clusters of SMP's are now attractive for executing high performance applications such as simulation applications. High performance and high availability are two desirable properties that need to be ensured for proper execution of long-running parallel applications on a cluster. We work on the design of Gobelins, a cluster operating system, to provide these two properties to parallel applications based on the shared memory programming model.

To achieve both high performance and high availability, we investigate an approach based on global integrated processor, memory and disk management in the operating system. Global resource management allows an application to use any cluster resource whichever node it is located in. A cluster is thus seen as a single machine rather than as a set of machines having their own resources. Although global management of memory, disk and processor have already been extensively studied, very few work has been done for global resource management in which these mechanisms *cooperate* or are *integrated* towards the objective of executing high performance applications. Moreover, very few work has been done on reliable global resource management.

Global and integrated resource management towards high performance and high availability is Gobelins main feature. Several advantages follow from this approach. Common mechanisms are only implemented once in the operating system. For example, process checkpointing and process migration mechanisms both require the extraction of a process state, the former for saving it in stable storage, the latter for sending it to a remote node over the network. Global resource management can also avoid to take conflicting decisions in the system. For example, taking into account both memory and processor loads in a process migration mechanism can avoid a negative impact of a process migration on the memory system (as would be the case if a process is migrated to a node executing only a single process with large memory requirements, the migrated process would generate many page faults). Finally, information known by a subsystem can be used to optimize another one. For example, the page state (modified or not) managed by a shared virtual memory can be exploited by a checkpointing mechanism to only save modified data when a process checkpoint is established.

As we consider shared memory parallel applications, Gobelins is based on a COMA-like software Distributed Shared Memory (DSM) providing both shared virtual memory and remote paging. It offers two supplementary subsystems both based on this DSM: one providing integrated memory and disk management and the other one dealing with integrated memory and processor management. Gobelins is designed to tolerate transient and a single permanent node failure.

Gobelins is a distributed system implemented as an extension of Linux. It is currently under development on a cluster of dual-processor PC interconnected by a low latency high bandwidth network.

This paper presents the main design features of Gobelins system. It is organized as follows. Background work on global resource management in distributed systems is presented in Section 2. Section 3 gives an overview of Gobelins operating system design. Global and integrated management of disk and memory resources is considered in Section 4. Section 5 is devoted to global and integrated management of processor and memory resources. Section 6 concludes and gives the current status of the experimental work.

2 Background

A lot of work has already been done in global resource management to exploit the characteristics of current clusters

such as low latency communication, huge primary memories and high speed processors. In such systems, nodes resources are managed in a single unified and distributed way instead of being managed independently and autonomously by each node. In this section, we briefly review previous work on global management of memory, disk and processor resources.

2.1 Memory Resource

Two kinds of mechanisms have been studied: remote memory paging and shared virtual memory systems.

Remote memory paging is an alternative to disk paging. The idea is to exploit the main memory of remote nodes instead of disk for paging [7, 11, 10]. Remote paging has been described in the context of Apollo DOMAIN system [14]. In [7], a remote memory model is proposed in which a cluster contains dedicated remote memory servers that can be used by nodes having heavy paging activity. This idea is generalized in [11] which proposes to use the memory of idle nodes as a paging device. In [9], reliable remote paging is studied but few work deals with fault tolerance issues related to remote paging.

The work described in [10] goes a step further by managing memory globally at the lowest level of the operating system. Thus virtual memory paging, mapped files and file system buffering are naturally integrated by using the low level global memory management algorithm. As only clean pages can be sent in global memory, node failures are tolerated.

Distributed shared virtual memory systems [16] are another way to manage memory in a cluster. It provides the illusion of a global shared memory on top of distributed memories. Node local memories are used as local caches of the shared data space. Shared data is migrated or replicated in the node memories. Many coherence protocols have been proposed to manage the coherence of multiple copies of the same data. Some research has been carried out to improve the reliability of DSM systems. However, few recoverable DSM described in the literature have been implemented [19].

2.2 Disk Resource

In contrast to distributed file systems that only distribute a file system among multiple file servers by partitioning the directory tree, parallel file systems offer global management of the disk resource. Several parallel file systems ([8, 24] distribute data over multiple storage servers to support parallel workloads. Except XFS [24], most of them lack mechanisms to provide high availability. Global disk management is also provided in Zebra striped network file system [12] designed to achieve high performance and high availability.

RAID-derived storage systems [22] provide high performance and highly available disk storage. However, software distributed management of parity may significantly hurt the performance.

XFS provides to some extent integrated disk and memory management by taking advantage of cooperative caching to harvest portions of client memory as a large, global file cache. Single level storage systems [14, 2] also provide some kind of integrated memory and disk management.

2.3 Processor Resource

Preemptive process migration [25] allows global management of the processor resource in a cluster. This mechanism may be supervised by load distributing algorithms to move a process from one node to another, in order to take advantage of available resources. However, few operating systems have implemented it [21, 23, 25, 20]. In most of these systems, each workstation is individually owned thus limiting global processor management. Migration mechanisms implemented in message-passing distributed systems are often limited to individual processes which do not communicate with others and are thus not suitable to parallel applications.

Some form of global and integrated processor and memory management is provided in Mosix through the memory ushering algorithm [3]. This algorithm is activated when a node's free memory falls below a threshold value and attempts to migrate processes to other nodes which have sufficient free memory. Thus, process migration is decided not only based on processor load criterion but also taking into account memory usage.

3 Gobelins Overview

Gobelins is based on a page-based COMA-like software DSM providing both shared virtual memory and remote paging. The nodes' memories are used as large caches. Pages, which represent the unit of transfer and coherence, are automatically migrated or replicated in the memory of the node of the processor which references them. Our system is based on a write-invalidate coherence protocol implemented by statically distributed directories. When a page has been loaded in memory, existing copies are used to serve subsequent page faults thus avoiding disk accesses. When a page needs to be replaced in a node local memory, it may be injected in the memory of a remote node or copied to disk locally or remotely depending on several parameters.

Gobelins offers two supplementary subsystems both based on Gobelins DSM: one providing integrated memory and disk management and the other one dealing with integrated memory and processor management. Integrating global memory and disk management has led to the design of a single level storage system [14] based on a Parallel File System (PFS). A PFS [8] is indeed desirable for the execution of high performance applications in a cluster as it provides increased disk bandwidth by fragmenting a file on several disks, allowing parallel accesses to the fragments. File mapping is the PFS interface thus releasing the programmer from explicitly managing page transfers between memories and disks. In addition, PFS caches and write buffers are merged with the DSM. Very few other work has been done on file mapping in software DSM with a PFS [17].

Integrated memory and processor management relies on process migration. A DSM transfers pages to the requesting processor node memory on demand. However, in some cases, it may be more efficient to migrate the process execution context (only comprising of the process registers and stack as data are assumed to be managed by the DSM) to the node holding the accessed pages. Gobelins integrating both a DSM and a process migration facility migrates either pages or processes depending on several factors including processor load and memory criteria (e.g. usage, contents).

4 Global and Integrated disk and memory Management

4.1 File mapping in DSM

Global memory and disk management in our system has led to the design of a one level storage system [14] built from a page-based Shared Virtual Memory (SVM) system

and a PFS. In contrast to the *read/write* interface which is traditional in PFS, file mapping in SVM is the natural interface of the PFS in the system we propose. This kind of interface has several advantages. File accesses are implicitly performed by memory operations. Hence, no parallel file pointer needs to be managed. The coherence of concurrent accesses to the same file are automatically managed by the SVM. Data being automatically loaded in the memory of the processor accessing them, the programmer does not have to manage data distribution. As PFS (client and disk) caches and the SVM are merged, their size is dynamically and automatically adjusted depending on data access patterns. The use of prefetching allows automatic overlap of computing and data loading. Programming parallel out of core applications is easier since files are simply viewed as another memory level.

Concerning the page replacement strategy, the idea is to keep at least a copy of a page in global memory as long as there is a chance that it is referenced by a processor. When page frames are needed in a node, the number of copies of a given page may be decreased or a page copy may be injected in a remote node. The disk paging mechanism implemented in most modern operating systems works by evicting pages from memory according to a Least Recently Used (LRU) selection scheme. In Gobelins, the selection scheme uses page state (of the coherence protocol) information to better select pages to evict from a local memory. Page replacement is selectively performed depending on these states.

Redundant copies of a page shared by several nodes have the highest priority of eviction. They are simply discarded on replacement. When the last copy of a page has to be replaced, it is injected in a remote node having memory space available. If there is no space available in global memory it is copied to disk. A page belonging to a mapped file is copied to its disk counterpart. Other pages allocated in the SVM are copied to the local swap disk.

4.2 Performance Issues

In traditional PFS, programmers can specify, through the *read/write* interface, data access pattern to help the operating system to optimize disk accesses. In the proposed system, file mapping which is the PFS interface makes disk accesses transparent to the programmer. A data is loaded from disk on a page fault. Disk writes may only occur in the event of a page replacement or at the end of the application. Thus, in our system, it is essential to optimize data loading performance. However, as the PFS does not have any information about the application data access pattern, optimization of disk accesses is more difficult than in a traditional system.

A first prototype of the proposed system has been developed on a cluster of PCs interconnected by a SCI network [4] and running Linux. Preliminary results obtained by performance measurements on this prototype, and not presented here for space reasons, show that the implementation of an efficient prefetching strategy is required [5]. So we are currently working on a mechanism to automatically find a close approximation of a parallel application logical data access pattern at execution time. This will allow our system to prefetch data, leading to increased performance. To achieve this automatic pattern detection, we investigate different approaches, in particular artificial neural networks and Markov chains [18].

Another problem related to file mapping as a new PFS interface is the scheduling of disk accesses. In our system, each thread issues a single disk read request at a time generated by a page fault. If several threads of the same application access simultaneously different areas of the same file located in a single disk, this leads to many disk seeks. A good scheduling policy should permit a low number of disk seeks without decreasing the potential computing/loading overlap.

4.3 Fault Tolerance Issues

In the proposed system, tolerating a single disk failure or a single node reboot or failure is essential as memory and disk management is distributed. We assume that nodes operate in a *fail-silent* fashion and that the interconnection network is reliable. The unit of failure is the node: the failure of a node component leads to the unavailability of the whole node.

To tolerate disk failures, mechanisms inspired from the RAID technology [6] are used. Gobelins PFS relies on a RAID-1 organization (also called mirroring). A file fragment (usually called a striping unit) is replicated on two disks. The only drawback of this organization is the disk capacity needed (twice the capacity of a non fault tolerant PFS). However, RAID-1 organization has several advantages. First, a page can be read from any of the two disks thus balancing the load on two nodes. Moreover, a page can be preferably read from the disk with the shorter queuing. Write operations imply writing the page on the two disks. If a disk fails, the other one can be used to serve all the requests in degraded mode thus resulting in a low performance degradation. Despite its cost in disk capacity, a RAID-1 strategy seems more adequate than a RAID-5 strategy in Gobelins which primarily aims at high performance. Indeed, a RAID-5 would be less expensive in term of disk capacity but much more complex to implement efficiently in distributed software [24]. Using a RAID-1 organization is all the more reasonable that current disks offer a high storage capacity at a low cost.

Tolerating node failure during the execution of long-running applications can be achieved using backward error recovery techniques [15]. Programmers of parallel scientific applications usually manage checkpoints manually by periodically saving the computation state. In the event of a breakdown, computation is restarted from the last checkpoint. When using a SVM, efficiently establishing a checkpoint at the application level is complex because of the lack of knowledge on data location in memories. In the proposed system implementing file mapping data location in memory and on disk is known by the system which can thus offer checkpointing primitives. Moreover, the mapping system has a detailed knowledge of data state (modified or not modified), allowing to save only modified data when a checkpoint is established [13]. Checkpoint data are saved in memory. When memory space is needed, they are in priority and efficiently copied to disk in a dedicated log-structured file system easing the garbage of data belonging to stale checkpoints.

5 Global and Integrated Processor and Memory Management

5.1 Scheduling strategies in parallel Architectures

A cluster constitutes a unique architecture for the execution of parallel and sequential applications. However, sharing the resources of a cluster between these two types of application while satisfying the performance requirements of parallel applications implies the design of adequate scheduling strate-

gies and efficient system mechanisms to implement them. Several scheduling policies have already been designed in parallel machines. Two main classes may be distinguished: space sharing and time sharing. With space sharing policies, processors of the machine are partitioned. Each partition executes a single application. Partitioning may static or dynamic depending on whether it is modified or not when a new application arrives in the execution queue. The draw back of space sharing policies is that a short duration application execution may be blocked during the execution of a long-running application.

Time sharing policies have also been designed for parallel machines. They are of two types: local scheduling and co-scheduling. Local scheduling strategies rely directly on local processor schedulers. Each processor has a single execution queue in which priorities are managed. Sequential processes are not distinguished from processes of a parallel application. This leads to many synchronization delays at the expense of parallel application performance. Co-scheduling strategies have been designed for parallel application scheduling. They allow several parallel programs to enter a service queue. Processes belonging to the same program are scheduled simultaneously on different processors for a given execution time, called a slot. At the end of a slot, all the processes of the parallel application are suspended to allow the execution of another parallel program. A drawback of co-scheduling strategies is that they only concern workloads that are exclusively composed of parallel applications.

5.2 Global Scheduling in Gobelins

New scheduling strategies are needed in clusters in order to satisfy the performance requirements of the various applications that may be executed. In Gobelins system, the processor resource is globally managed following an hybrid strategy between space sharing and time sharing policies to satisfy the needs of both sequential applications (minimizing the response time) and parallel applications (minimizing the execution time taking into account synchronization constraints) while supporting their concurrent execution on a cluster. With its global scheduling strategy and its process migration mechanism, Gobelins offers the image of a shared memory scalable multiprocessor.

The implementation of a global scheduling policy in a cluster is based on the realization of process migration mechanism. Such a mechanism has been studied in message-passing distributed systems where it is motivated by the need to optimize processor or network load. Hence, migrating a process implies the transfer of the whole process context which is extremely expensive for large processes. In Gobelins, the process migration mechanism is integrated with the DSM. The process state is divided in two parts: the execution context and the data set. The execution context comprises of the process stack and registers. The data set comprises of the process private and shared data and code. The data set is managed by the Gobelins subsystem integrating the DSM and the PFS. Thus, data is loaded on demand in the execution context node. The data set may be distributed in several nodes due to data sharing or page replacement. In Gobelins migration mechanism, only the execution context of a process is migrated. Its data set is progressively and automatically migrated or replicated during its execution by the DSM.

However, in some cases, it may be more efficient to migrate a process to a node which memory contains the data it currently accesses rather than moving this data in the memory of its execution node (this would generate page faults which are expensive operations in term of performance in a DSM). Gobelins takes into account the *affinity* of a process with a node memory [1]. Suitable policies based on the joint observation of the cluster memory and processor states have still to be defined to decide of the best location of a process at a given time, knowing that the system offers both a process execution context migration mechanism and a DSM.

5.3 Fault Tolerance Issues

High availability is one of the main design goals of Gobelins. In order to tolerate a node failure during the execution of an application, Gobelins provides a process checkpointing mechanism. Such a mechanism is similar to a process migration mechanism in that both mechanisms need to extract the current process state: to save it in stable storage in checkpointing, to transfer it to a remote node in migration. However, these two mechanisms differ in that processes need to coordinate when checkpointing to ensure that a global system state is saved.

On another point of view, many previously proposed migration mechanisms rely on residual dependencies [25]. When a process is migrated a residual dependency remains on the process source node in order to facilitate the management of communications and I/Os involving the migrated process. Such residual dependencies are not desirable in Gobelins as a migrated process should be able to continue its execution despite the failure of its source node. Moreover, a process which is restored after a failure should be able to restart its execution on any node of the cluster despite the fact that the faulty node may be its source node.

As previously stated in Section 4.3, Gobelins provides a recoverable DSM integrated with a fault tolerant PFS. A process checkpointing mechanism is integrated with the recoverable DSM. As Gobelins recoverable DSM deals with process data set checkpointing, only a process execution context checkpointing mechanism needs to be added. When a checkpoint is established, not only the memory state is checkpointed but also the process execution context within a global two-phase commit protocol.

A process execution context checkpoint is called a process checkpoint in the remainder of this section. When a process checkpoint is established two copies of the execution context are saved: one in the process execution node called *ec-ckpt1* and another one in a remote node called *ec-ckpt2*. To balance the load of keeping process checkpoints, the *ec-ckpt2* copy is saved in the neighbor node of the process execution node, assuming a logical ring.

In the event of a failure of node N, each process has to be restored. Each node restores the execution context of processes for which it stores the *ec-ckpt1* copy of the checkpoint. In the event of a permanent failure of node N, each node holding the *ec-ckpt2* copy of a process checkpoint for which the *ec-ckpt1* copy is on the faulty node N is also responsible for restoring the corresponding process.

6 Conclusion

In this paper, we have introduced the main design features of Gobelins, a cluster operating system. Gobelins aims at providing high availability and performance to sequential and parallel applications executed on a cluster by an approach based on *global and integrated* management of proces-

sor, disk and memory resources. Gobelins is currently being developed as an extension of Linux. Experimentations are carried out on a cluster of dual-processor PC interconnected by a SCI high bandwidth network.

Acknowledgements

The author would like to thank Françoise André and Maria-Teresa Segarra for their contribution on global memory and processor management and Thierry Priol and Renaud Lottiaux for their contribution on global memory and disk management.

References

[1] F. André, C. Morin, and M.-T. Segarra. Mechanisms for global processor and memory management on a NoW. In *Proceedings of the HPCN Europe'98 International Conference*, pages 324–336, Amsterdam, The Netherlands, April 1998. LNCS 1401.

[2] J.P. Banâtre and M. Banâtre, editors. *Les systèmes distribués : l'expérience du projet Gothic*. InterEditions, February 1991.

[3] A. Barak and A. Braverman. Memory ushering in a scalable computing cluster. In *Proc. of IEEE third Int. Conf. on Algorithms and Architecture for Parallel Processing*. IEEE, December 1997.

[4] P. Beaugendre, T. Priol, G. Allion, and D. Delavaux. A client/server approach for HPC applications within a networking environment. In *HPCN'98*, LNCS 1401, pages 518–525, April 1998.

[5] A. Hayzelden J. Bigham, editor. *Software Agents for Future Communications Systems*, chapter Mobile Agents for Managing Networks: the Magenta perspective. Springer verlag, 1999. to appear.

[6] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[7] D. Comer and J. Griffioen. A new design for distributed systems: The remote memory model. In *Proc. of the USENIX Summer Conference*, pages 127–135, June 1990.

[8] J. M. del Rosario and A. Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.

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

[10] M.J. Feeley, W.E. Morgan, F.H. Pighin, A.R. Karlin, and H.M. Levy. Implementing global memory management in a workstation cluster. In *Proc of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[11] E.W. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical Report 91-03-09, University of Washington, March 1991.

[12] J. Hartman and J. Ousterhout. The zebra striped network file system. pages –, Asheville, North Carolina, December 1993.

[13] A.-M. Kermarrec, C. Morin, and M. Banâtre. Design, implementation and evaluation of ICARE: an efficicent recoverable DSM. *Software Practice and Experience*, 28(9):981–1010, July 1998.

[14] P. J. Leach, P. H. Levine, B. P. Douros, J. Hamilton, D. L. Nelson, and B. L. Stumpf. The architecture of an integrated local network. *IEEE Journal on Selected Areas in Communications*, SAC-1(5):842–856, November 1983.

[15] P.A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*, volume 3 of Dependable Computing and Fault-Tolerant Systems. Springer Verlag, second revised edition, 1990.

[16] K. Li and P. Hudack. Memory coherence in shared memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.

[17] S. C. Mac, C. K. Shieh, J. C. Ueng, and L. M. Tseng. Design and implementation of a parallel file subsystem on treadmarks. In *Proc. of the International Computer Symp.*, pages 256–263, December 1996.

[18] Tara M. Madhyastha and Daniel A. Reed. Exploiting global input/output access pattern classification. In *Proceedings of SC97: High Performance Networking and Computing*. ACM Press, November 1997.

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

[20] F. Douglis J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software Practice and Experience*, 21(8):757–785, 1991.

[21] M.L. Powell and B.P. Miller. Process migration in demos/mp. In *Proc. of the 9th Symposium on Operating Systems*, pages 110–119, 1983.

[22] Michael Stonebraker and Gerhard A. Schloss. Distributed RAID — A new multiple copy algorithm. In *Proceedings of 6th International Data Engineering Conference*, pages 430–437, 1990.

[23] M.M. Theimer, K.A. Lantz, and D.R. Cheriton. Preemptable remote execution facilities for the V-System. In *proceedings of the 10th Symposium on Operating Systems Principles*, pages 2–12, December 1985.

[24] 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.

[25] A. Barak S. Guday R. Wheeler. *The MOSIX Distributed Operating System*, volume 672 of *LNCS*. Springer Verlag, 1993.