Efficient Process Migration based on Gobelins Distributed Shared Memory

Geoffroy Vallée, Christine Morin, Jean-Yves Berthou, Ivan Dutka Malen, Renaud Lottiaux

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Efficient Process Migration based on Gobelins Distributed Shared Memory

Geoffroy Vallée, Christine Morin, Jean-Yves Berthou, Ivan Dutka Malen, Renaud Lottiaux

N°4518

Juillet 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4518--FR+ENG

Efficient Process Migration based on Gobelins Distributed Shared Memory

Geoffroy Vallée*, Christine Morin†, Jean-Yves Berthou‡, Ivan Dutka Malen§, Renaud Lottiaux¶

Thème 1 — Réseaux et systèmes
Projet PARIS

Rapport de recherche n ° 4518 — Juillet 2002 — 12 pages

Abstract: Clusters are attractive for executing sequential and parallel applications. However, there is a need to design cluster distributed operating system to provide a Single System Image. A cluster operating system providing both a DSM system and load balancing is attractive for efficiently executing a workload of sequential applications and shared memory parallel applications . Gobelins is a distributed operating system dedicated to clusters that provides both a DSM system and a process migration mechanism to support load balancing. In this paper, we present the implementation of Gobelins process migration mechanism which exploits Gobelins kernel level DSM system. We show that Gobelins DSM allows to implement simply an efficient migration mechanism, that can be used to move processes or threads among cluster nodes. A prototype of Gobelins has been implemented. Some performance results are presented in this paper.

Key-words: process migration, distributed shared memory, distributed operating system

(Résumé : tsvp)

* IRISA/Université de Rennes1/EDF/RESAM
† IRISA/Université de Rennes1
‡ EDF R&D
§ EDF R&D
¶ IRISA/INRIA (postdoc industriel co-financé par EDF)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Migration de processus efficace sur la mémoire répartie partagée du système Gobelins

Résumé : Les grappes de calculateurs sont attractives pour exécuter des applications parallèles ou séquentielles. Néanmoins, il est nécessaire de disposer pour cela d'un système d'exploitation distribué pour grappe qui permette de créer un système à image unique. Un système d'exploitation pour grappe offrant une Mémoire Partagée Répartie (MPR) et un équilibrage dynamique de charge est attractif pour exécuter efficacement des applications séquentielles ou parallèles à mémoire partagée. Gobelins est un système d'exploitation dédié aux grappes de calculateurs offrant à la fois une MPR et un mécanisme de migration de processus pour supporter un équilibrage de charge. Dans ce document, nous présentons la mise en œuvre du mécanisme de migration de processus qui exploite la MPR noyau de Gobelins. Nous montrons que la MPR de Gobelins permet de mettre en œuvre simplement un mécanisme de migration efficace, qui peut être utilisé pour déplacer des processus ou des threads sur les différents nœuds de la grappe. Un prototype de Gobelins a été mis en œuvre. Une étude de performance préliminaire est présentée dans ce document.

Mots-clé : migration de processus, mémoire paratagée répartie, système d'exploitation distribué

1 Introduction

Clusters are attractive for executing both sequential and parallel applications that are data and/or compute intensive. There are two common parallel programming paradigms: message passing and shared memory. In the message passing model, programmers explicitly manage data exchanges between processes. In the shared memory model, which is generally recognized as a simpler programming model than the former one, threads communicate through memory sharing using synchronization primitives.

The message passing programing model is generally supported in clusters by runtime environments such as PVM [?] or MPI [?]. The shared memory model is supported by Distributed Shared Memory (DSM) systems that are most of the time implemented at user level [?, ?] although some DSM systems are implemented at kernel level to take benefit of specific hardware mechanisms [?, ?, ?, ?].

Cluster operating systems have to be designed to *efficiently* execute workloads made up of sequential and parallel applications based either on the message passing or the shared memory programming paradigms. Such systems should provide transparent access to distributed resources, allow memory sharing among processes and threads and optimize resource utilization by implementing load balancing. However, cluster operating systems generally only support the message passing programming model [?]. To our knowledge, Genesis [?] is the only cluster operating system that offers a kernel level support for both programming models.

Gobelins is a cluster operating system we design to provide both a Distributed Shared Memory (DSM) system and a load balancing mechanism for efficiently executing a workload composed of sequential applications and parallel applications based on the shared memory paradigm. Gobelins provides a DSM system to support the shared memory paradigm and a process migration mechanism to support load balancing, both implemented at kernel level. In this paper, we present the implementation of Gobelins process migration mechanism which exploits the kernel level DSM system. Gobelins DSM not only allows data sharing among threads of a parallel application but it is also the foundation of global memory management in the cluster. Hence, any process takes advantage of the DSM to exploit the cluster memory as a remote paging device or a cooperative file cache.

We show that Gobelins DSM allows to easily implement an efficient migration mechanism. Gobelins process migration can be easily implemented as the existing DSM system is used to transfer the address space of a migrated process on demand. Moreover, as the file cache is based on the DSM, a migrated process can access open files on its new execution node in the same way as on its originating node. Gobelins process migration is efficient. First, the time to transfer a process state is kept low and constant as the address space is transferred on demand. Second, in contrast to Mosix[?] in which processes cannot take benefit of their execution node file cache after migration, accesses to open files in Gobelins are performed efficiently by a migrated process thanks to the use of the DSM as a cooperative cache. Finally, Gobelins is based on Linux. In Linux system, threads are implemented as processes. So, the Gobelins process migration is generic as the same mechanism is used to move independent processes or threads of a shared memory parallel application among

cluster nodes. A prototype of Gobelins operating system has been implemented integrating both a DSM system and a process migration mechanism. Preliminary performance results show that Gobelins process migration is efficient.

The remainder of this paper is organized as follows. In Section 2, we review previous work on process or thread migration in DSM systems. We present Gobelins operating system focusing on its DSM system in Section 3. The implementation of Gobelins process migration is described in Section 4. Performance results are analyzed in Section 5. Section 6 concludes.

2 Background

The shared memory model is an efficient and simple programming model and is offered by DSM systems on clusters. In some DSM systems, that we call *process-oriented* DSMs, a parallel application is implemented as a set of processes. Each application process corresponds to an operating system process. In other DSM systems, that we call *thread-oriented* DSMs, a parallel application is implemented as a set of threads. In this case, all the application threads that execute on a particular node belong to a single operating system process. We review in this section previous work on process or thread migration in the context of DSM systems.

User-level thread-oriented DSM systems usually rely on a multi-threaded run-time as in DSM-PM2 [**?**], D-CVM [**?**] or Millipede [**?**]. In DSM-PM2 and Millipede, the thread memory space is reserved within all the nodes, and so if a thread migrates, it can find its own virtual memory space anywhere. In D-CVM system, a thread can be migrated only after all the application global variables have been initialized and under restrictive constraints. In fact, some system calls cannot be executed by a migrated thread. These systems provide a very efficient thread migration mechanism but which cannot be used for existing applications based on processes. Moreover, thread-oriented DSM systems limit the application size to a given number of threads due to the memory allocation within all the nodes.

There is not a lot of *process-oriented* DSMs that offer process migration. To our knowledge, only Genesis [**?**] and Startdust[**?**] allows to migrate processes. Stardust is an environment for parallel programming on networks of heterogeneous workstations which provides a load balancing mechanism for heterogeneous systems. Genesis is a system based on a microkernel (all the distributed system services are created from scratch), which provides some distributed mechanisms like process (or group of processes) creation, and a DSM service. In this system, a process or a group of processes using the DSM can migrate. Genesis also provides a global file management: a file access is independent of the process location. So, a process which accesses files can migrate anywhere within the cluster. However, Genesis doesn't have a thread runtime, so existing multi-threaded application cannot be directly executed on top of Genesis.

3 Overview of Gobelins

Gobelins is an operating system aiming at giving a single system image of a cluster. Gobelins has been designed to efficiently execute multi-threaded applications. Gobelins federates all resources available within the cluster performing global management of disks, memories and processors. Gobelins can be implemented by slight modifications of an existing single node operating system. Our first prototype is based on the Linux kernel. In Linux, a thread is implemented as a process. Linux system (as any modern system) can be divided in two layers : a virtual layer and a physical layer. The virtual layer implements the application system interface and resources visualization. The physical layer implements devices access (hard disk, memory). Physical memory used by a process is mapped in its virtual memory. Virtual memory mapping allows two processes to share some memory segments (clone processes share their text segment for example). Containers are inserted between these two layers (see figure 1). So, every system event between the virtual and the physical layers can be intercepted by containers, allowing to divert and to widen traditional operating system services, and to access distributed data.

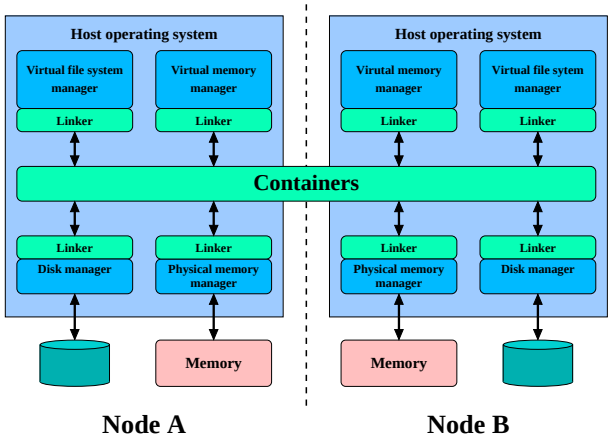

Figure 1: Architecture of containers

Gobelins is based on the Linux kernel. In the Linux kernel, and also in the Gobelins kernel, threads are implemented by processes. So, a process migration mechanism can be used to migrate threads : it's possible to implement a generic migration mechanism both for processes and threads.

To offer the vision of a SMP machine, Gobelins extends traditional system mechanisms thanks to the container concept. A container is a software object which allows to store and share memory pages between cluster nodes. Each container is associated with two linkers : a interface linker which diverts high level kernel functions to containers, and an i/o linker

which allows containers to access to a given device manager. For each distributed system service, two different linkers are used. For example, we can implement a DSM using a linker to connect a container to the virtual memory manager, and another one to connect a container to the physical memory manager. In this case, a container can be seen as an extension of the memory segment concept to the cluster scale. The process address space is divided into memory segments which can be associated to a container (see figure 2), and can be shared between different processes.

With the container mechanism, a DSM, a cooperative file cache and a distributed file system are implemented in the system.

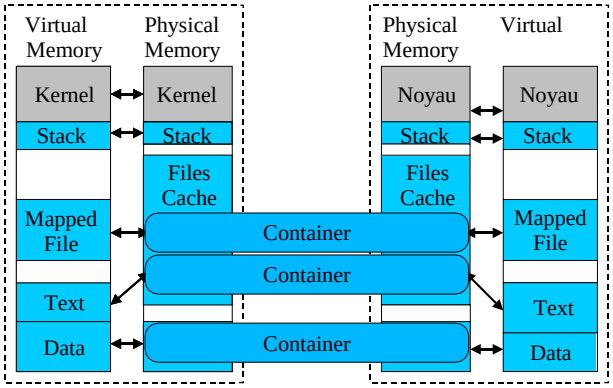

Figure 2: Data sharing with containers in Gobelins

Gobelins allows to execute standard Linux applications as well as Gobelins applications.

4 Design of Gobelins Process Migration

The process migration consists of three phases. The first one is the extraction and transfer of process information from the process execution node (source node). The second one is the information reception, and the process restart on another node chosen for the migration (destination node). The third one is the execution of the migrated process.

In the remainder of this section, we first present the extraction and the transfer of process information. We present in a third part issues related to process execution after migration.

4.1 Extraction of Process Information for Migration

In Gobelins operating system (like in the Linux system), a process is a set of information: the process address space, the opened files list, the processor registers state and the process stack. Most of these informations are available from the kernel structure of each process

(**task_struct**). The two most difficult issues for process migration are the migration of the process address space and the access to opened files after migration. In this paper, we focus on both the memory and the file issues and show how they can be solved using Gobelins DSM.

There are two kinds of memory management data: on one hand the page table which allows to solve paging requests, on the other hand some kernel structures to manage physical memory pages.

The kernel structure **mm_struct** manages the general memory information. The memory segments are managed by the **vm_area_struct** and all these segments (linear addresses) are associated with physical memory pages. In Gobelins, some **vm_area_struct** are linked with a container (see Figure 3). These links allow to access remote memory pages (so, implement the Gobelins DSM service).

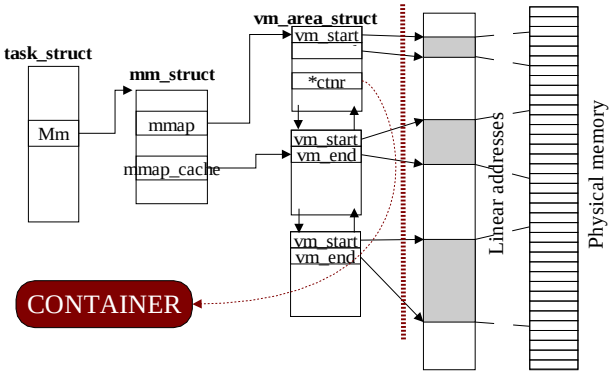

Figure 3: Kernel data structures of a process virtual memory with containers

Kernel file structures used for the management of opened files are more complex. We only provide a brief overview of file kernel management in the Gobelins kernel.

There are three different file structures: **inode**, **file** and **dentry**. The **inode** structure allows to physically access a file on disk. But several processes can simultaneously access the same file. So, processes manipulate **file** structures instead of **inode** structures. The **dentry** structure is used to implement symbolic links.

The kernel also manages two kinds of linked lists: the first one contains all files opened in the system while the second one contains files opened by each process.

Moreover, some memory segments are linked with a file. In Gobelins, to provide access to remote files a virtual inode and a virtual dentry linked to a container are created: every file data managed by a container is available all over the cluster (see Figure 4) through Gobelins DSM.

So, the Gobelins DSM simplify the extraction of process informations for migration. For the memory information, we just need to extract **mm_struct**, **vm_area_struct** and

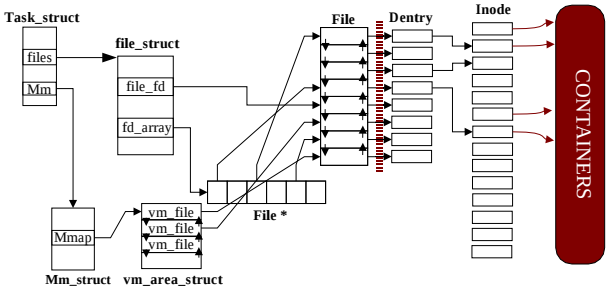

Figure 4: Kernel data structures of a process open file list with containers

container identifier. We don't care about page tables and physical pages migration. For file memory information, we just need to extract **file** kernel structures and container identifier. We don't care about **dentry** and **inode** structures.

4.2 Transfer

Gobelins container mechanism allows the implementation of a DSM service over the cluster. Pages stored in the DSM do not need to be migrated during the process transfer. Only container identifiers need to be transferred in order to rebuild the link between memory segments (or files) and containers (calling the **link_vma_to_container** Gobelins kernel function) on the destination node. This function creates the interface linker and links a **vm_area_struct** with a container, which allows to use DSM services. So, for migration of memory information, we just need to migrate **mm_struct** and **vm_area_struct** information, and we don't care about the migration of physical memory pages and about the building of the page table. All physical memory pages and virtual address space can be accessed through the Gobelins DSM.

The container mechanism allows to share files accessed within the cluster through the DSM. So, for the process migration, we don't need to migrate every data structures representing files accessed by a process. As for memory data, we just need to migrate high level open files data structures (**file** kernel structure) and container identifiers. With these data we can rebuild the link between high level kernel data structures and containers.

So, we just have a few informations to transfer. We only need to transfer high level informations about memory and file accessed by a process. These informations are kernel structures which not depend on the process memory size or on the number a accessed files. So, the size of informations to transfer is constant, and not depend on the size of the process.

4.3 Execution of a Migrated Process

Once a process is migrated, it can resume his execution. Containers have then two advantages.

First, memory pages available in DSM (so in containers) and accessed by the process are copied on destination node on demand, only when accessed. So, if the process is a sequential process, only pages accessed by the process are migrated on the destination node. If it's a thread (or a process) of a parallel application, it can always access shared data through the Gobelins DSM mechanism.

Second, when a process access a file, data are loaded in the system file cache. These memory pages are available through the DSM, and it is so possible to access it from another nodes anywhere in the cluster. If any process accesses these pages (the migrated process or another one), Gobelins allows to send them to the process through the DSM. So, containers allow to take advantage of a cooperative file cache at the cluster level. If the process accesses a file which is not yet in the cooperative cache (a file which has not yet been accessed for example) and which is not locally present on the node, containers allow to make a remote file access.

So, processes can take benefit in one hand of the DSM service to access remote memory page, but also to benefit of a cooperative cache through the Gobelins DSM, which allows to efficiently access to files.

5 Evaluation

In this section, we only present a preliminary evaluation of the migration mechanism described in previous sections used in the context of sequential applications executed on top of Gobelins. In this first evaluation, we show how Gobelins DSM makes process state extraction and transfer efficient as well as the process execution after migration. We first evaluate the cost of a process transfer in Gobelins and second, we evaluate the execution overhead of a migrated process.

5.1 Experimental Platform

The experimental platform consists of two nodes based on Intel Pentium Pro 200MHz processors with 128 MB physical memory and interconnected with a Fast Ethernet network.

Gobelins operating system is an extension of Linux version 2.2.13. Currently, Gobelins is implemented in three modules: a module implementing high performance kernel to kernel communications, a module which implements containers, and a module which implements process management.

5.2 Implementation Details

To be efficient, the migration mechanism has to comply to a set of constraints. A process migration should be possible at any time during execution. Moreover, when a process

migration is decided, the process kernel data structure should be available in a consistent state. Thus, the process migration mechanism is implemented as an internal operating system mechanism, transparent to users. It is implemented in the Gobelins system with a few kernel modifications. The process migration mechanism is based on the Linux kernel function **do_fork** (kernel function to perform a fork system call) to restart the process on the destination node.

A migration activation system call has been added in the system. This system call marks the process as being waiting for migration. The process is actually migrated only when the system is in an adequate state in which all the needed information regarding the process state is available.

To integrate the migration mechanism, the kernel has been slightly modified to create a new state, traced on the signal treatment (less than 50 lines modified in the kernel to create this state). Every time the system exits from an interrupt or a system call treatment, the task waiting to resume its execution is analyzed to determine if it requires a migration or not.

Two different migration mechanisms have been implemented in our prototype: one for the migration of Gobelins processes (processes which use the Gobelins DSM) and one for the migration of traditional Linux processes (for which the whole process address space is migrated). The former is called *Gobelins migration* and the latter is called *basic migration* in the remainder of this paper. For these two mechanisms, a kernel daemon is implemented on each node to receive at any time data transferred when a process is migrated. When all data has been received, this daemon resumes the process calling the **do_fork** function.

For our evaluation, a sequential application executing the Modified Gram-Schmit (MGS) algorithm has been used. This algorithm produces from a set of vectors an orthonormal basis of the space generated by these vectors. The algorithm consists of an external loop running through columns producing a normalized vector and an inner loop performing for each normalized vector a scalar product with all remaining ones. This program is written in C and does not include any special feature to perform the migration.

We made a set of preliminary measurements on various sizes of the matrix used in MGS algorithm. A migration is triggered at an arbitrary time during the application execution. This allows us to quantify the impact of the migration depending on the size of the application (i.e. the application data set size).

The application migration is always triggered manually, using the migration system call. When an application is created, it is a standard Linux process which does not use the Gobelins DSM. During the migration initialization the migration mechanism links containers to the application memory segments. All the migration time measurements are made using kernel functions. The process execution time is computed in the kernel with process time information when the process finishes its execution (in the kernel function **do_exit**).

We first compare the migration cost for a Gobelins process and the migration cost for a standard process. So, we evaluate the migration time for different matrix size in these two cases. Table 1 shows that the migration cost for standard migration is proportional with the number of memory pages used, while the migration cost over the Gobelins DSM is constant

Matrix Size	With containers (without initialization)	Without containers
500x500	83.54	205.57
750x750	83.46	240.45
1000x1000	83.31	283.39
1250x1250	83.34	319.51

Table 1: Migration cost (in msec)

(about 83 ms). This time is due to the transfer time, made sending a lot of little messages, and so, the transfer is not efficient. With the containers, the migration cost does not depend to the number of memory pages used by the process: the memory pages are transfered after the process migration, during the process execution, on demand. So, the migration cost of the memory pages is included in the time of the process execution, after migration, reducing the inactive time during migration. In the case of a standard migration, all the memory pages are migrated during the process migration and so the migration cost increase with the number of pages used.

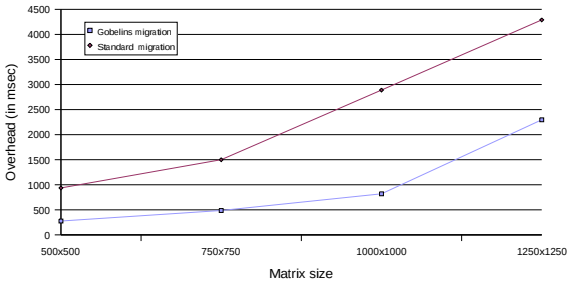

Figure 5: Migration overhead on process execution time

Now, it is interesting to evaluate the migration overhead on process execution time. Indeed, once the application is migrated, memory pages in containers need to be migrated on the destination node on demand. To do that, we evaluate the execution time of process which migrates once on different matrix sizes.

Figure 5 shows the migration overhead on the total application execution time considering a process which migrates once during the application execution either using Gobelins or the basic migration mechanism.

Results show that Gobelins migration is more effective than the basic migration mechanism. The overhead due to the DSM management is lower than the overhead incurred by the transfer of the whole process address space.

6 Conclusion

In this paper, we have presented Gobelins process migration mechanism which is based on a kernel level DSM system. Gobelins DSM is exploited by the process migration mechanism: memory pages need not be transfered when a process is migrated and are only migrated using the standard DSM mechanisms if accessed by the migrated process after migration. Gobelins DSM also provides a cooperative file cache implemented by its DSM system which is exploited by the process migration mechanism. This feature allows efficient accesses by migrated process to all files (local or remote, open before migration or not). This is one of the main contributions of this paper.

A first evaluation of Gobelins migration mechanism shows that migrating a process in Gobelins is more expensive than in thread-oriented system like DSM-PM2. However, Gobelins migration mechanism is generic as the same mechanism is used to migrate a process of a process-oriented or a thread of a thread-oriented parallel application. We have shown that the process migration mechanism based on Gobelins DSM is more efficient than a process migration mechanism that does not take benefit of the DSM and transfers the whole process memory space at migration time. Moreover, Gobelins process transfer cost is kept constant due to the use of the DSM system.

Gobelins migration mechanism allows to easily implement other system services such as a *checkpoint/restart* facility for sequential and parallel applications or a *remote process creation* mechanism which makes easy the deployement of parallel applications on clusters. All these mechanisms have in common with a process migration mechanism the management of the process image.When an application is checkpointed, the process image is stored in memory or on disk. In remote process creation, the process image is duplicated to create processes on remote nodes (this mechanism is already implemented in our prototype). We currently work on the implementation of a checkpoint/restart mechanism based on the process migration mechanism described in this paper and which relies on the protocols described in [**?**]. We also work on the design and implementation of a global scheduler

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399