A Framework for High Availability Based on a Single System Image

Geoffroy Vallée, Christine Morin, Stephen L. Scott

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Framework for High Availability Based on a Single System Image

Geoffroy Vallée , Christine Morin , Stephen L. Scott

N°5734

Octobre 2005

Systèmes numériques

ISSN 0249-6399 ISRN INRIA/RR--5734--FR+ENG

A Framework for High Availability Based on a Single System Image

Geoffroy Vallée* , Christine Morin† , Stephen L. Scott‡

Systèmes numériques
Projet PARIS

Rapport de recherche n˚5734 — Octobre 2005 — 10 pages

Abstract: High availability (HA) is today an important issue in the domain of cluster computing, clusters being more and more larger, introducing a lot of failures. Today, the literature provides a lot of different HA strategies to tolerate application failures (applications being sequential or parallel). Unfortunately, it is still difficult to implement these HA policies inside a real system, and therefore the study of these policies is most of the time just theoretic, without real implementation.

Therefore, a framework to ease the implementation of such policies is interesting. Moreover, a single system image (SSI), thanks to mechanisms for the global management of cluster resources, is a good candidate to provide such a framework.

This paper presents the preliminary study of this framework on top of the Kerrighed SSI.

Key-words: high availability, operating system, single system image

(Résumé : tsvp)

* valleegr@ornl.gov - ORNL/INRIA/EDF - Computer Science and Mathematics Division - Oak Ridge National Laboratory - Oak Ridge, TN 37831, USA. INRIA Postdoc co-funded by EDF R&D and ORNL

† cmorin@irisa.fr - IRISA/INRIA, PARIS project-team - Campus Universitaire de Beaulieu, 35042 Rennes, Cedex, France

‡ scottsl@ornl.gov - ORNL - Computer Science and Mathematics Division - Oak Ridge National Laboratory - Oak Ridge, TN 37831, USA

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

Un environnement pour la haute disponibilité fondé sur un système á image unique

Résumé : La haute disponibilité est aujourd'hui un problème important pour les grappes de calculateurs, ceux-ci ayant une taille de plus en plus grande, introduisant de nombreuses fautes. Pour cela, la littérature offre de nombreuses stratégies permettant de tolérer les fautes d'applications (que les applications soient séquentielles ou parallèles). Malheureusement, la mise en œuvre de ces politiques de haute disponibilité est toujours difficile et leur étude est donc très souvent limité à une étude théorique, sans réelle mise en œuvre.

Un environnement dédié simplifiant la mise en œuvre de telles politiques est donc intéresant. De plus, un Système à Image Unique (Single System Image - SSI), grâce à ses mécanismes de gestion globale des ressources de la grappe, est un bon candidat pour offrir un tel environnement.

Ce document présente l'étude préliminaire d'un tel environnement fondé sur le SSI Kerrighed.

Mots-clé : tolérance aux fautes, système d'exploitation, système à image unique

1 Introduction

Clusters are today widely used as computing resource thanks to effective systems. To guarantee good performances, cluster's systems also need to provide high availability (HA) features in particular for long running and critical applications. Some systems already provide HA policies or HA mechanisms [3, 8], some of them guaranteeing high availability for applications and others for systems services.

For example, HA-OSCAR [4] is a software suite which allows to deploy and manage a cluster with high availability mechanisms for the head node. With such a solution, the head node is duplicated and if a failure occurs on one of these head nodes, another one can replace it without interruption of cluster services.

On another hand, some mechanisms provide high availability for applications. For example, BLCR [2], EPCKPT [12] provide checkpoint/restart of applications. With such mechanisms, if a failure occurs on a compute node the system is able to restart applications without to have to restart from scratch.

Based on these systems, it is possible to have HA clusters. Nevertheless, current systems often provide basic HA policies (like synchronized application checkpoint, active-passive headnode duplication). At the same time the literature provides a large set of HA policies but without the possibility test them on real systems and therefore without the possibility to evaluate their benefit.

This paper presents a framework to implement HA policies for applications. This framework is based on the Single System Image (SSI) Kerrighed, SSI providing a large set of distributed services needed to implement HA policies such as process checkpoint/restart, global file system and so on.

The remainder of this paper is organized as follows. Section 2 gives an overview of the Kerrighed SSI. Section 3 presents the framework for the implementation of HA policies. Section 4 presents some implementation details and finally Section 5 concludes.

2 Overview of the Kerrighed Single System Image

Single System Image (SSI) aims at providing a global management of distributed resources in cluster. With such a global resource management, a cluster can be view as an SMP machine. Therefore, a SSI globally and transparently manages cluster resources. SSI aims at providing a support for both parallel and sequential programming paradigms, (*e.g.* MPI, OpenMP). For that, SSI systems provide basic mechanisms like process migration, process checkpoint/restart, software distributed shared memory (DSM), and global data streams management (mechanisms for migration and checkpoint/restart of sockets, pipes and signals). Finally, users can execute applications, taking advantage of distributed resources, without application modifications. SSI also aims at managing node failures

[1]ORNL, Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA, {valleegr, scottsl}@ornl.gov.

[2]INRIA, Campus Universitaire de Beaulieu, 35042 Rennes, Cedex, France, cmorin@irisa.fr

with high availability mechanisms. The goal is to be able to tolerate node failures, guaranteeing application and system execution.

Today, several project aims at providing a SSI for Linux clusters. The three major projects for Linux SSI are openMosix [15], OpenSSI [11] and Kerrighed [10, 9, 18]. Each of these projects take a different approach for creating a SSI. Because of these different approaches, performance for each system differs depending on the application [7].

OpenSSI is the integration of existing projects to provide a complete solution for global resource management. OpenSSI allows a user to globally manage processes, disks and data streams. However the global memory management is not supported.

OpenMosix is a SSI focused on high performance through a global scheduling mechanism to guarantee efficiency for application execution. OpenMosix does not provide a global memory management, nor an efficient migration of data streams (like sockets), because of a dependency with the original node when a migration occurs. High availability issues are out the scope of the openMosix project.

Kerrighed is a SSI, developed from scratch that extends the Linux kernel, which globally manage all resources: disks, memories, data streams, and processors. An advantage of Kerrighed is to be able to setup the system according to an application's needs. For the global management of processes, Kerrighed provides mechanisms for process migration, process checkpoint/restart, process duplication and process remote creation. All these mechanisms are based on a common mechanisms for process virtualization [16]. This common mechanism, called *ghost process*, ease the maintenance of mechanisms for global process management and also ease the implementation of new mechanisms. Moreover, ghost processes can be used with other mechanisms, such as global memory management, allowing the extension of thread management of the traditional Linux kernel.

Therefore, SSI and in particular Kerrighed is an interesting platform for the implementation of high level services (like load balancing and HA) thanks to mechanisms for global resource management.

3 Framework for the Implementation of High Availibility Policies

SSI provides a large set of mechanisms for global resource management. These mechanisms may be used to implement policies which are needed to offer high level services (like thread support across compute nodes or HA policies for parallel applications).

Unfortunately, each kind of applications (*e.g.* sequential applications, parallel applications using a shared memory, parallel applications by message passing) has different needs. For example for a parallel application, a checkpoint/restart policies has to deal with dependencies between processes/threads [3, 8] (in the rest of the document, we speak about processes of parallel applications even for threads of a parallel application using a shared memory at the cluster scale). If this policy cannot manage dependencies, the application checkpoint cannot be coherent. Moreover, the literature provides a large set of different policies, some of them being specialized for a specific kind of applications.

The checkpoint/restart of sequential applications is quite simple: the process of the application has to be checkpoint and this image allows to restart the process on a similar machine (a machine for which checkpointed data are pertinent). Therefore, for the checkpoint/restart of sequential applications, information about memory, IO have to be checkpointed. These informations being available in the kernel, the implementation of such a mechanism is not complex and is available in several systems.

The checkpoint/restart of parallel applications (by message passing or shared memory) is much more complex because of the lack of global information in the checkpoint of a single process. In the checkpoint of a single process, no information about communications between processes of a parallel application are saved. Therefore, it can be impossible to restart the application. To avoid that, we need to checkpoint dependencies between processes in order to have a global coherent checkpoint of the application. Based on that, two major approaches are possible: (i) synchronization of processes before the checkpoint or (ii) recovery to a global coherent state during the restart thanks to additional checkpointed information. The synchronization of processes before the checkpoint is simple to implement. We just need to create a barrier between processes and when all processes are in the barrier, the global state of the application is coherent, it is possible to checkpoint the application. The advantage of this approach is to allow a quick restart (all processes are synchronized, they can be restarted being sure of the application coherency) but the checkpoint is slow because of the global synchronization. The synchronization of processes during the restart step avoids this synchronization cost. Processes can be checkpointed without any synchronization, only information about communications are also checkpointed (for example a log of communications). During the restart, the system tries to create a global coherent state of the application thanks to process checkpoints and communication logs, finding a *recovery line* [1, 5]. With this approach, the checkpoint is simple and fast but in some cases, the restart can be very expensive because of the time to create a recovery line (in some cases, it is impossible to create a recovery line, the application has to be restarted from scratch; this effect is called the domino effect [13, 14]).

Therefore, to implement and to use these policies, a framework for the implementation of HA policies on top of cluster system is interesting. This HA framework is also very similar to a framework to implement scheduling policies: the framework needs to provide (i) a mechanism which allows to change the policy dynamically without interruption of system services or applications, (ii) a development kit to ease the development of HA policies.

3.1 Development Kit for HA Policies

To implement an HA policy, the developer needs first to implement the algorithm of the policy. Studying traditional algorithms available in the literature, a set of primitive has been identified to implement HA policies.

We saw that to implement an HA policy, we need two kind of primitives: (i) primitives to checkpoint/restart a process and (ii) primitive to manage communications between processes of a parallel application.

Analyzing HA policies, four kind of primitives has been identified to checkpoint/restart processes. First of all, a primitive to checkpoint a process. This primitive allows to extract a process

and to save this extracted image (on disk for example). Then, restart primitives allow to restart a process from a checkpoint. For that, two approaches are possible: (i) restart a process creating a new process in the system, or (ii) restart a process rolling back an existing process. Finally, for parallel applications, synchronization primitives allow to synchronize a set of processes of a parallel application or the complete application.

For the management of dependencies between processes of a parallel application (communications by message passing or memory pages sharing), traditional checkpoint/restart mechanisms have to be extended. For parallel application using a shared memory, the checkpoint/restart mechanism has to checkpoint the shared memory (memory pages and system information of the DSM system). For example, in Kerrighed the shared memory is based on *containers* [6], the checkpoint/restart mechanism has to manage containers information to guarantee a global coherency of the parallel application. It is the same issue for parallel application based on the message passing paradigm. The checkpoint/restart mechanism has to manage sockets in order to guarantee that sockets are still in a coherent status after a restart even if the process is restarted on another node.

The extension of the checkpoint/restart mechanism is not sufficient because of the possible domino effect. The log of events related to dependencies between processes of a parallel application is also needed. Therefore, a primitive has to be available to log each events related to the DSM (*i.e.* page fault, copy of a memory pages from a node to another or page invalidation) or network communications (*i.e.* sent or received messages).

This set of primitives ease the implementation of HA policies but the implementation of HA policies for clusters is still quite complex: the algorithm has to be distributed to avoid any single point of failure, the mechanism to execute the policy on compute nodes has to tolerate faults.

3.2 Mechanism for the Dynamic Plug of HA Policies

To allow the dynamic modification of the HA policy, without interruption of applications or system services, the system needs to provide a mechanism to plug and unplug the policy inside the system.

For that a possible approach is to use a mechanism to dynamically load system components (which implement the HA algorithm) based on a description mechanism to hide system details for its use. Therefore, a new tool based on Kerrighed and kernel modules has been created. This tool is composed by tree components (see Figure 1): (i) the XML description of the policy, (ii) the tool to generate kernel modules from the XML description and (iii) the loader which loads generated kernel modules. Thanks to this approach, to implement a new policy, the programmer can use a high level toolkit which hides all technical points due to the implementation at the system level of the policy. The only constraint for the developer is to program a distributed algorithm that is compliant with the distributed architecture of the Kerrighed tool (there is no global configuration, the configuration is only local; if the policy needs a global view of the cluster, the programmer has to implement it in the algorithm, *e.g.*, the diffusion of information about compute node has to be integrated in the HA algorithm).

The XML description allows to ease the description of the policy thanks to a specific XML schema.

Finally, technical details of the system programming are hidden.

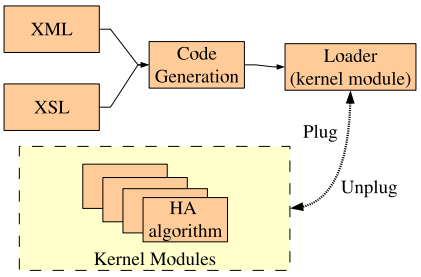

Figure 1: Mechanism for Hot Plug of Clustering Policies

4 Implementation Details

Kerrighed is already used as platform for the implementation of system policies adapted to application needs, using a development framework. The development of a framework for HA policies is composed of two parts: (i) the development of system mechanisms for HA (*e.g.* process checkpoint/restart) and (ii) the development of of the framework itself to ease the programing of HA policies.

4.1 Mechanisms for High Availability

Kerrighed provides various mechanisms for HA: process checkpoint even for processes that use sockets or the Kerrighed's distributed shared memory and process restart for the same kind of processes. The process rollback is not yet finalized.

All these mechanisms are based on a single mechanism called *ghost processes* [17] (see Figure 2). This common mechanism allows to virtualize processes (the ghost process mechanism al-

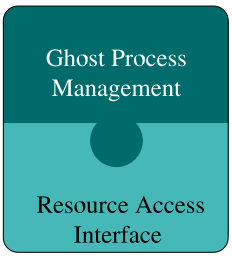

Figure 2: Ghost Process Architecture

lows to create an image of a process that can be used to create a new clone process in the cluster independently to the initial location of the process) and allows to specify which resource to use for

the process virtualization. The current Kerrighed version allows for example to checkpoint/restart process on disk and in memory.

4.2 Framework for the Implementation of HA Policies

The framework for the implementation of HA policies is partial: mechanisms for the dynamic plug of HA policies is finished but the development kit is still in development.

4.2.1 Development Kit for HA Policies

The development kit for HA policies is not yet finalized. However, a framework for the development of scheduling policies is available. This framework allows to ease the implementation of various scheduling policies and provides a large set of primitives.

For HA policies, primitives for checkpoint/restart (but not for process rollback) and some primitives for process synchronization (Kerrighed provides internal mechanisms for process synchronization) are available. To ease the development of new policies, a GUI can be used by the programmer (see Figure 3). The GUI provides primitives for HA policies but also a set of macro that hides sys-

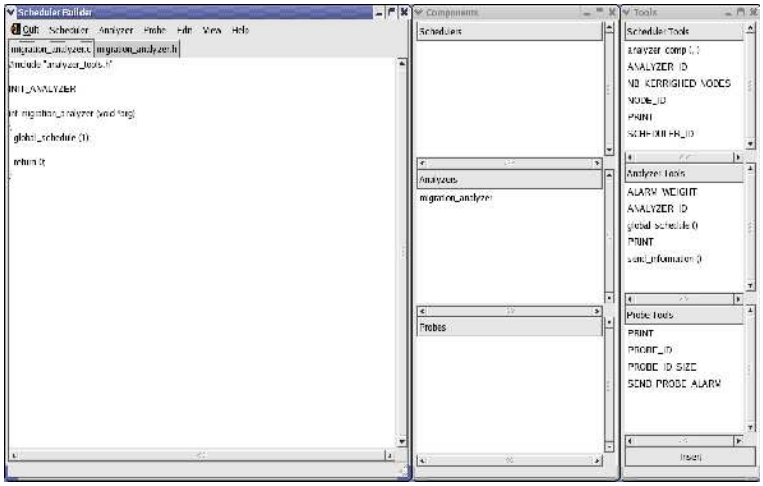

Figure 3: Framework GUI for the Development of Clustering Policies

tem programing details. The code created using the GUI is then compiled in kernel modules. The dynamic property of kernel modules allows to guarantee the possibility to plug/unplug a policy in the system, and technical issues of system programming are hidden.

Primitives for the log of events to track DSM events and network events are not available. However, Kerrighed provides a mechanism for checkpoint/restart of MPI applications but this mechanism has not been implemented thanks to the framework for HA policies.

4.2.2 Mechanism for the Dynamic Plug of HA Policies

Kerrighed provides a mechanism for the dynamic plug of system policies. This mechanism is currently used for the implementation of load balancing policies, which allows to dynamically change the scheduling policy. This mechanism is used to load the global scheduler in Kerrighed.

The dynamic plug of system policies is based on the dynamic property of kernel modules: the XML description allows to automatically generate a kernel module with can plug and unplug kernel modules implementing the HA policy created using the development kit (GUI and primitives).

The mechanism is not yet used for an HA framework.

5 Conclusion

Today the literature provides a large set of different HA policies for various kind of applications. Therefore, a framework for the implementation and the dynamic plug of HA policies is interesting.

SSI, thanks to mechanisms for global management of resources, is a good solution to provide such a framework for the implementation of HA policies.

The development of this framework is still in development but a framework for the implementation of scheduling policies, very closed to the framework for the implementation of HA policies, is already available. To have a complete framework for the implementation of HA policies, some primitives are still missing but does not seam complex to implement.

The finally framework will ease the implementation of HA policies and Kerrighed will be a good platform to evaluate and the compare of HA policies.

References

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

[2] J. Duell, P. Hargrove, and E. Roman. The design and implementation of berkeley lab's linux checkpoint/restart. Technical report, Berkeley Lab, 2003.

[3] M. Elnozahy, L. Alvisi, Y-M. Wang, and D.B. Johnson. A survey of rollback-recovery protocols in message-passing systems. Technical Report CMU-CS-99-148, Carnegie Mellon University, June 1999.

[4] Ibrahim Haddad, Chokchai Leangsuksun, and Stephen L. Scott. Ha-oscar: the birth of highly available oscar. *Linux J.*, 2003(115):1, 2003.

[5] Richard Koo and Sam Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Trans. Softw. Eng.*, 13(1):23–31, 1987.

[6] R. Lottiaux and C. Morin. Containers : A sound basis for a true single system image. In *IEEE International Symposium on Cluster Computing and the Grid, May 2001.*, 2001.

[7] Renaud Lottiaux, Benoit Boissinot, Pascal Gallard, Geoffroy Vallée, and Christine Morin. Openmosix, openssi and kerrighed: A comparative study. In *Cluster Computing and Grid 2005 (CCGRID 2005)*, Cardiff, England, May 2005.

[8] C. Morin and I. Puaut. A survey of recoverable distributed shared memory systems. *IEEE Trans. on Parallel and Distributed Systems*, 8(9):959–969, 1997.

[9] Christine Morin, Pascal Gallard, Renaud Lottiaux, and Geoffroy Vallée. Towards an efficient Single System Image cluster operating system. *Future Generation Computer Systems*, 20(2), January 2004.

[10] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, Gaël Utard, Ramamurthy Badrinath, and Louis Rilling. Kerrighed: a single system image cluster operating system for high performance computing. In *Proc. of Europar 2003: Parallel Processing*, volume 2790 of *Lect. Notes in Comp. Science*, pages 1291–1294. Springer Verlag, August 2003.

[11] openSSI.org. *Introduction to the SSI Cluster*. http://www.openSSI.org/.

[12] Eduardo Pinheiro. Truly-transparent checkpointing of parallel applications.

[13] Brian Randell. System structure for software fault tolerance. *Software Engineering*, 1(2):221–232, 1975.

[14] D. L. Russell. State restoration in systems of communicating processes. *IEEE Trans. Software Eng.*, SE-6(2):183–194, March 1980.

[15] Esposito Mastroserio Tortone. Openmosix approach to build scalable hpc farms with an easy management infrastructure.

[16] Geoffroy Vallée, Renaud Lottiaux, David Margery, Christine Morin, and Jean-Yves Berthou. Ghost process: a sound basis to implement process duplication, migration and checkpoint/restart in linux clusters. In *The 4th International Symposium on Parallel and Distributed Computing*, Lille, France, July 2005.

[17] Geoffroy Vallée, Renaud Lottiaux, David Margery, Christine Morin, and Jean-Yves Berthou. Ghost process: a sound basis to implement process duplication, migration and checkpoint/restart in linux clusters. In *The 4th International Symposium on Parallel and Distributed Computing*, Lille, France, July 2005.

[18] Geoffroy Vallée, Renaud Lottiaux, Louis Rilling, Jean-Yves Berthou, Ivan Dutka-Malhen, and Christine Morin. A case for single system image cluster operating systems: Kerrighed approach. *Parallel Processing Letters*, 13(2), June 2003.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
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