inria-00346955, version 1 - 12 Dec 2008

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

The Architecture of the XtreemOS Grid Checkpointing Service

John Mehnert-Spahn — Thomas Ropars — Michael Schoettner — Christine Morin

N° 6772

Décembre 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6772--FR+ENG

inria-00346955, version 1 - 12 Dec 2008

inria-00346955, version 1 - 12 Dec 2008

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

The Architecture of the XtreemOS Grid Checkpointing Service

John Mehnert-Spahn* , Thomas Ropars†‡ , Michael Schoettner* , Christine Morin†§

Thème NUM — Systèmes numériques
Équipe-Projet PARIS

Rapport de recherche n° 6772 — Décembre 2008 — 19 pages

Abstract: The EU-funded XtreemOS project implements a grid operating system (OS) transparently exploiting distributed resources through the SAGA and POSIX interfaces. XtreemOS uses an integrated grid checkpointing service (XtreemGCP) for implementing migration and fault tolerance. Checkpointing and restarting applications in a grid requires saving and restoring applications in a distributed heterogeneous environment. The latter may spawn millions of grid nodes using different system-specific checkpointers saving and restoring application and kernel data structures on a grid node. In this paper we present the architecture of the XtreemGCP service integrating existing checkpointing solutions. Our architecture is open to support different checkpointing strategies that can be adapted according to evolving failure situations or changing application requirements. We propose to bridge the gap between grid semantics and system-specific checkpointers by introducing a common kernel checkpointer API that allows using different checkpointers in a uniform way. Furthermore, we discuss other grid related checkpointing issues including resource conflicts during restart, security, and checkpoint file management. Although this paper presents a solution within the XtreemOS context it can be applied to any other grid middleware or distributed OS, too.

Key-words: Grid Computing, Checkpointing, Fault Tolerance

This research is supported by XtreemOS project under European Commision FP6 Contract (No. 033576).

* Department of Computer Science, Heinrich-Heine University, Duesseldorf, Germany. John.Mehnert-Spahn, Michael.Schoettner@uni-duesseldorf.de
† PARIS project-team
‡ University of Rennes 1, Rennes, France. Thomas.Ropars@irisa.fr
§ INRIA Rennes-Bretagne Atlantique, Rennes, France. Christine.Morin@inria.fr

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

inria-00346955, version 1 - 12 Dec 2008

Architecture du service de sauvegarde de points de reprises dans XtreemOS

Résumé : Le projet européen XtreemOS a pour objectif de fournir un système d'exploitation de grille offrant un accès transparent aux ressources de la grille au travers des interfaces SAGA et POSIX. Au sein d'une grille XtreemOS, la migration d'applications et la tolérance aux fautes sont fondées sur un service de sauvegarde de points de reprise appelé XtreemGCP. Dans une grille composée d'un grand nombre de nœuds, les mécanismes disponibles pour sauvegarder l'état des applications diffèrent selon les nœuds. Dans cet article nous présentons l'architecture du service XtreemGCP. Cette architecture permet d'intégrer différentes solutions de sauvegarde de points de reprise disponibles sur un nœud et fournit différents protocoles de sauvegarde de points de reprises pouvant être adaptés en fonction de l'état du système. Pour faire le lien entre les concepts utilisés á l'échelle de la grille et ceux utilisés par les systèmes d'exploitation classiques, nous proposons une API générique pour les systèmes de sauvegarde de point de reprise au niveau noyau. Cette API permet d'utiliser ces systèmes de manière uniforme. Dans cet article nous traitons aussi d'autres difficultés relatives à la sauvegarde de points de reprise dans les grilles comme les problèmes de conflits d'identifiants au redémarrage des applications, les problèmes liés à la sécurité ou encore la gestion des fichiers de points de reprise. Les solutions décrites dans cet article sont présentées dans le cadre d'XtreemOS, mais sont suffisamment génériques pour pouvoir être appliquées dans d'autres systèmes de grille.

Mots-clés : Grille, Sauvegarde de points de reprise, Tolérance aux fautes

inria-00346955, version 1 - 12 Dec 2008

1 Introduction

Grid[8] technologies can help to run businesses in dynamic distributed environments effectively. Grids enable/simplify the secure and efficient sharing and aggregation of resources across administrative domains. Grids may spawn millions of nodes and thousands of users.

Several sophisticated grid middleware systems emerged during the last years, e.g. Globus [7], Legion [11], and UNICORE [22]. Most of these systems were started with a bottom-up premise constructing services and tools according to changing user requirements. As a result there are various versions with different services and interfaces and the programmer needs to spend considerable time on basic Grid functions.

The key idea of the EU-funded XtreemOS project is to reduce the burden on the Grid application developer by providing a Linux-based open source Grid operating system (OS). XtreemOS provides a sound base for simplifying the implementation of higher-level Grid services because they can rely on important native distributed OS services, e.g. security, resource, and process management [1]. In XtreemOS a grid node may be a single PC, a LinuxSSI cluster (SSI = Single System Image), or a mobile device. The LinuxSSI version is an extension of the Linux-based Kerrighed system [25] providing the illusion of a cluster appearing as one single powerful grid node. Because of resource constraints mobile nodes act as clients rather than fully fledged grid nodes.

Grid technologies offer a great variety of benefits but there are still challenges ahead including fault tolerance. With the increasing number of nodes more resources become available but at the same time the failure probability increases. Fault tolerance can be achieved for many applications using a rollback-recovery strategy. In the simpliest scenario an application is halted and a checkpoint is taken that is sufficient to restart the application in the event of a failure. Beyond using checkpointing for fault tolerance it is also a basic building block for application migration, e.g. used for load balancing.

A lot of different checkpointing strategies have been proposed in the literature how to efficiently checkpoint (distributed) applications. Although checkpoint strategies are very important within a grid environment they are beyond the scope of this paper. Here we focus on the XtreemOS checkpointing service (XtreemGCP) architecture aiming at integrating existing checkpointing technologies.

There are numerous state of the art system-specific checkpointing solutions supporting checkpointing and restart on single Linux machines, e.g. BCLR [14], Condor [16], libCkpt [20]. They have different capabilities regarding what kind of resources can be checkpointed. Furthermore, there are also specific distributed checkpointing solutions for cluster systems dealing with communication channels and cluster-wide shared resources, like the Kerrighed checkpointer [25].

Of course it is not realistic to select one of these implementations and to enforce its use on each grid node. Instead we aim at integrating existing checkpointing solutions by introducing a common kernel checkpointer API that is implemented by customized translation libraries. Therewith, different checkpointing packages can be used in a uniform way. Due to the dynamic characteristics of grids XtreemGCP must also be able to adapt checkpointing parameters and strategies during runtime.

The contribution of this paper is to propose an architecture for the XtreemGCP service integrating different existing checkpointing solutions. As a proof of concept this architecture has/is implemented within the XtreemOS project which is available as open source [1]. Although the concept and ideas described in this paper are presented in the context of XtreemOS they can also be adapted to any other grid system.

The outline of this paper is as follows. In Section 2 we briefly present relevant background information on XtreemOS, use cases and requirements. Subsequently, we present the architecture of XtreemGCP. In Section 4 we discuss the common kernel checkpointer API and implementation aspects. Other grid related issues, e.g. resource conflicts and security are described in Section 5. Related work is discussed in Section 7 followed by conclusions and future work.

2 Checkpointing an Application in XtreemOS

Because XtreemGCP is fully integrated with different XtreemOS services we briefly describe the relevant ones in this section. Subsequently, we present use cases followed by requirements for the XtreemGCP architecture.

2.1 XtreemOS Terminology and relevant Services

In this paper, we consider sequential, parallel, and distributed applications. We call a job an application that has been submitted to the grid. Each job has a grid-wide unique job id. A job is divided into job-units, a job-unit being the set of processes of a job running on one grid node.

Applications are handled by the application execution management service (AEM) [5]. AEM job managers handle jobs life cycle, including submission, execution and termination. Each job is managed by one job manager. A job directory service stores the list of active jobs and the location of their associated job managers. On each grid node, an execution manager handles the job-units running on the node. It performs the actions requested by the job manager and is in charge of controlling and managing job-units, applying signals and monitoring them.

XtreemOS includes a grid file system called XtreemFS [12], providing location-transparent file access and file replication and striping. In this paper, XtreemFS is used as persistent storage for grid checkpoint files.

2.2 Use Cases

Checkpoint/restart is of major importance in XtreemOS - it is the basic block for several functionalities: (i) *taking a job snapshot*, which is the traditional usage of checkpoint mechanisms; (ii) *job migration*, which is the sequence of job checkpointing, job killing and job restarting on destination grid node(s); (iii) *job suspension*, that consists of job checkpointing, job killing and restarting on the same or on other grid nodes at a later time.

In XtreemOS, these functionalities are used for:

- Scheduling: job migration is used to optimize scheduling. A job can be moved to other nodes for load balancing reasons or to allow other jobs to

inria-00346955, version 1 - 12 Dec 2008

be executed. A job can also be suspended, if a higher priority job has to be executed first, and be restarted afterwards.

- Node disconnection: Before disconnecting a node from the grid, the node owner should notify AEM. Once the jobs running on this node have been migrated/suspended the node may be disconnected from the grid.
- Fault tolerance: To avoid computation loss in case of failures, job checkpoints are taken periodically. If AEM detects a failure, the job is restarted using the before saved checkpoint data. At job submission, the user specifies whether job fault tolerance is to be applied or not. XtreemGCP assumes a fail-stop failure model for nodes.
- Debugging: Checkpointing can also be useful to debug applications. Using checkpoints, an application can be run in the past in order to localize lingering mistakes that are hard to detect.

2.3 Grid Checkpointing Requirements

Subsequently, we describe generic requirements for a grid checkpointing service

- *Consistency*: At restart, a checkpointed application must be restored in a consistent global state.
- *Scalability*: Grids allow users to execute large scale distributed applications. A checkpoint action can cause significant overhead because of large checkpoint image network transfers or process synchronization.
- *Transparency*: XtreemGCP should be able to checkpoint jobs without the need to modify or recompile an application before.
- *Heterogeneity*: A grid is heterogeneous regarding hardware and software dimensions. XtreemOS grid nodes can be PCs or clusters each of them coming with one or several kernel checkpointers. XtreemGCP must act as mediator between these kernel checkpointers to checkpoint applications spread over heterogeneous grid nodes.
- *Various rollback-recovery protocols*: must be supported to allow XtreemGCP to apply the best suited rollback-recovery protocol for each application.
- *Adaptation*: A grid environment as well as the application behaviour may change over time. Thus, XtreemGCP must adapt its job checkpointing strategy to reflect these changes and checkpoint in an efficient manner. Furthermore, the lack of application-internal knowledge causes the service to checkpoint the whole application content each time, including for example large data files, possibly unnecessarily. The application developer and the user must be given the option to adapt checkpointing by defining what actually needs to be checkpointed.
- *Security*: Rollback-recovery mechanisms can induce security problems. Users rights may have changed between checkpoint and restart. A library used by the application may have been updated to solve a security hole. Such issues have to be solved at restart. Generally, only authorized users,

inria-00346955, version 1 - 12 Dec 2008

inria-00346955, version 1 - 12 Dec 2008

i.e. the application owner or the AEM on behalf of the application owner, should be able to checkpoint and restart an application.

- Ease-of-use: Management of checkpoints should be as simple as possilbe. XtreemGCP should be able to automatically restart a failed application or to restart it on demand from a specific checkpoint.

In the next sections, we describe how XtreemGCP fulfills these requirements.

3 Architecture of XtreemGCP

The XtreemGCP service architecture, its components and their interaction are explained here in the context of checkpointing and restarting grid applications.

3.1 Grid Checkpointing Services

The XtreemGCP is a layered architecure, see Figure 1. At the grid level, a job checkpointer manages checkpoint/restart for one job possibly distributed over many grid nodes. Therefore, it uses the services provided by the job-unit checkpointer available on each grid node. A job-unit checkpointer controls the kernel checkpointers available on the grid nodes to take snapshots of the job-units processes. The so-called *common kernel checkpointer API*, introduced by XtreemOS, enables the job-unit checkpointer to address any underlying kernel checkpointer in a transparent way. A translation library implements this API for a specific kernel checkpointer and translates grid semantics into specific kernel checkpointer semantics. In the following we provide details for all these services.

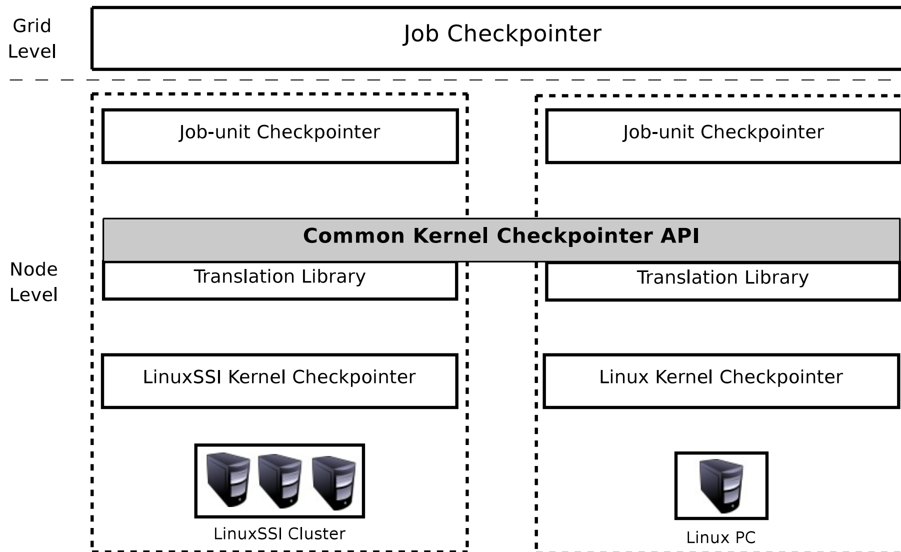

Figure 1: XtreemOS Grid Checkpointing Service Architecture

inria-00346955, version 1 - 12 Dec 2008

3.1.1 Job Checkpointer

There is one job checkpointer service per job in the grid. It is located on the same node as the corresponding AEM job manager. Furthermore, it distributes the load, induced by XtreemGCP over the whole grid. We use virtual nodes [21] to make job checkpointers and job managers highly available via service replication.

The job checkpointer has a global view of the job. It knows the job-units composing the job and the job-units location. It controls the underlying components to apply a checkpoint decision. When checkpointing is used to migrate or suspend a job, the decision is taken by the AEM. In case of checkpoints used for fault tolerance, the job checkpointer is responsible for scheduling the checkpoints.

In coordinated checkpointing, the job checkpointer takes the role of the coordinator. With the help of the job-unit checkpointers, it synchronizes the job-units at checkpoint time to guarantee a consistent checkpoint. At restart, it coordinates the job-units to ensure a consistent job state, see Section 3.2.

Because of its global view on the job, the job checkpointer is in charge of creating job checkpoint meta-data. This meta-data describes the job checkpoint image. The contained information is required at restart to recreate a job-unit, see Section 5.2.

3.1.2 Job-Unit Checkpointer

There is one job-unit checkpointer per grid node. It interacts with the job checkpointer to apply a checkpoint decision. The job-unit checkpointer is in charge of checkpointing job-units. Therefore, it relies on the common kernel checkpointing API to transparently address a specific kernel checkpointer.

3.1.3 Common Kernel Checkpointer API

Each grid node comes with heterogenous kernel checkpointers having different calling interfaces, checkpoint capabilities, process grouping techniques, etc. One grid node may also offer several kernel checkpointers. Therefore, we have specified the *common kernel checkpointer API*, see Section 4, that allows the job-unit checkpointer to uniformly access different kernel checkpointers.

One important feature of this API is the possibility to register callbacks functions. Callbacks allow users to execute specific code at checkpoint and restart time. They can be used to deal with issues that are not handled by the kernel checkpointer or for optimizations, see Section 4.3.

A translation library implements the common kernel checkpointer API for one kernel checkpointer. Such a kernel checkpointer-bound translation library is dynamically loaded by the job-unit checkpointer when the kernel checkpointer is required to checkpoint/restart a job-unit. In general the translation library converts grid semantics into a kernel checkpointer semantics (e.g grid job IDs are translated into IDs that reference process groups).

3.1.4 Kernel Checkpointer

A kernel checkpointer is able to take a snapshot of a process group, including process memory and diverse process resources, and is able to restart the pro-

inria-00346955, version 1 - 12 Dec 2008

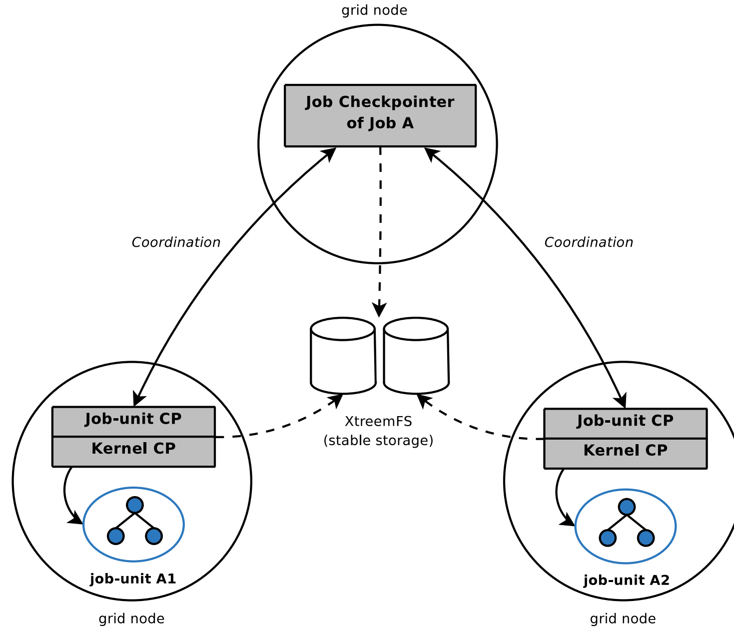

Figure 2: XtreemOS Grid Checkpointing Component Interaction

cess group from this snapshot. Unlike checkpointers exclusively implemented in user space, a kernel checkpointer is also capable of restoring kernel structures representing process' resources like process ids or session ids.

The *suspend* capability of virtual machines can also be used to checkpoint/restart processes [15]. Our service architecture supports usage of virtual machines as kernel checkpointer.

XtreemOS provides a default kernel checkpointer for each grid node. A checkpointer based on BLCR is available on the single PCs and the LinuxSSI kernel checkpointer based on the Kerrighed checkpointer is available on LinuxSSI nodes.

3.2 Checkpointing Strategies

In this section we present how our architecture supports different checkpointing strategies including coordinated and uncoordinated checkpointing. Here we only focus on the interactions between the services. Other issues related to checkpointing, like security or checkpoint file management, are described in Section 5.

3.2.1 Coordinated Checkpointing

Coordinated checkpointing is the default protocol used by the service used to realise job migration and suspension. The interaction of XtreemGCP components is shown in Figure 2.

inria-00346955, version 1 - 12 Dec 2008

Job checkpoint Upon receiving the job snapshot signal, the job checkpointer enters the *prepare phase*. In coordinated checkpointing, the job takes the role of the coordinator. It addresses those job-unit checkpointers residing on the same grid nodes as the job-units of the job to be checkpointed. The involved job-unit checkpointers load the appropriate translation library whose associated kernel checkpointer is capable of checkpointing the job. A job is notified of an upcoming system-initiated checkpoint action to prepare itself, e.g. an interactive application notifies the user to be temporarily unavailable. Then, the job-unit checkpointers extract the restart-relevant meta-data, described in Section 5.2.2, with the help of the kernel checkpointers storing this meta-data in XtreemFS.

Afterwards the job checkpointer instructs the job-unit checkpointers to enter the *stop phase*. Processes of all involved job-units are requested to execute their checkpoint-related callbacks, see 4.3. After having finished with these checkpoint-related callbacks, each job-unit process is stopped from further execution and an acknowledgement is sent by the job-unit checkpointers to the job checkpointer. Meta-data are updated in case processes have been created between initial saving and process synchronization.

The job checkpointer waits for all job-unit checkpointers acknowledgment to enter the *checkpoint phase*. The kernel checkpointers are requested through the job-unit checkpointers to take snapshots of the processes. These snapshots are saved onto XtreemFS. Afterwards the *resume phase* is entered.

Job restart At restart the *rebuild phase* is entered. The job checkpointer parses the checkpoint meta-data of the job to be restarted. This data is used by the job manager, to reallocate those resource IDs being valid before checkpointing, and the job-unit checkpointers, to select the same kernel checkpointers as the ones used at checkpoint time. Afterwards, the job checkpointer informs all involved job-unit checkpointers of the job-units belonging to the job that is to be restarted. Each job-unit checkpointer gets the job-unit checkpoint image it needs from XtreemFS and rebuilds the job-unit processes with the help of the kernel checkpointer. Subsequently, the job-unit checkpointers enter the *resume phase*. Here all restart callbacks are processed and afterwards the job-units resume with normal execution.

3.2.2 Uncoordinated Checkpointing

In uncoordinated checkpointing, checkpoints of job-units are taken independently. Thus, synchronization overhead at checkpoint time is avoided and the job-units decide when to checkpoint. The decision to trigger checkpoints is shifted from the job checkpointer to the job-unit checkpointers. They can take node state and job-unit state-information into account to decide when it is opportune to checkpoint, e.g. regarding the amount of data to be saved. The common kernel checkpointer API also allows an application to checkpoint a job-unit.

At restart, a consistent global state of the application has to be computed. Therefore dependencies between checkpoints have to be logged during failure-free execution. As described in Section 4.4, the AEM service monitors the processes to provide information about their communications. The involved job-unit checkpointers are in charge of analyzing this data in order to compute the dependencies between checkpoints. At restart, the job checkpointer uses the

dependency data saved with the checkpoints to compute the consistent global state. Then, it informs the appropriate job-unit checkpointers to perform the restart procedure.

The main drawback of uncoordinated checkpointing is the domino effect. The latter can be avoided, by using message logging. The job-unit checkpointer is in charge of saving messages for its job-units.

3.2.3 Application-level Checkpointing

In application-level checkpointing, the application itself executes the checkpoint and recovery functions. The checkpointing code is inside the application. This code is written directly by the programmer, added through code instrumentation or by providing a runtime environment or library. In case of application-level checkpointing, the application interacts with the job-unit checkpointer(s).

MPI has become a de-facto standard for high performance computing applications. Several MPI libraries provide fault-tolerance mechanisms. Some MPI libraries make use of system-level checkpointing in application-level checkpointing. Open MPI [13] and LAM/MPI [23] have chosen to implement generic checkpoint/restart mechanisms that can support multiple existing kernel checkpointers. Since this approach fits well with ours, these two libraries can be supported by our service, too. When an MPI library is used, the responsibility to coordinate the processes or to detect dependencies is shifted to this library. On the grid node where the *mpirun* has been launched the job-unit checkpointer merely initiates checkpointing and manages the checkpoint images created.

3.3 XtreemOS Adaptive Checkpoint Policy Management

Since the grid is a dynamic environment and applications consume resources in an unpredictable manner, XtreemGCP must be able to dynamically adapt its checkpointing policy for efficiency reasons.

As described before, XtreemGCP supports several checkpointing strategies. Both, coordinated and uncoordinated checkpointing, have pros and cons. Adapting checkpointing parameters or switching from one strategy to another depends on various factors, e.g. failure frequency, job behaviour, etc. Strategy changes can improve the performance, see [6].

The job checkpointer, job user and/or the application itself control switching between checkpointing strategies to avoid incompatible strategy-related snapshot data. Each job checkpointer collects monitoring data in a staged manner. Therefore, each job-unit checkpointer acquires job-unit activity and grid node states, analyzes them, forwards significant information to the job checkpointer to support the adaptation process.

However, job-unit checkpointers can independently vary checkpointing parameters within a given checkpoint strategy, e.g. changing checkpointing frequency, switching to/from incremental or concurrent checkpointing, change the checkpoint-image replication-level.

4 Common Kernel Checkpointer API

As said before we cannot assume that each grid application or grid node uses the same kernel checkpointer. The *common kernel checkpointer API* bridges differ-

inria-00346955, version 1 - 12 Dec 2008

ences to allow the job-unit checkpointers to transparently and uniformly access these different kernel checkpointers. The following sections gives an overview of the API and discusses important functionalities implemented within translation libraries.

4.1 API Description

The *common kernel checkpointer API* is implemented by the dynamically loadable translation library per kernel checkpointer. The current version of the implementation of the API provides following primitives:

- *xos_prepare_environment:* A process group is looked for which encapsulates a job-units processes and which is understood by a kernel checkpointer. In case of successful return the process group reference id and type are transferred to the job checkpointer to be saved in the checkpoint meta-data. The checkpoint sequence can be proceeded with.
- *xos_stop_job_unit* In order to call kernel checkpointer functionality, appropriate libraries are initialized (e.g. BLCR library). The translation library is able to address user space components and kernel components of diverse checkpointers. Subsequently, the execution of checkpoint callbacks is enforced. Afterwards, all processes belonging to the job-unit are synchronized.
- *xos_checkpoint_job_unit* A process group checkpoint is taken and saved in a XtreemFS volume. Thus the checkpoint image is grid-wide accessible in a location-transparent manner.
- *xos_resume_job_unit_cp* The processes are waked up in order to proceed with normal execution.
- *xos_rebuild_job_unit* The previously stored checkpoint meta-data indicate the appropriate XtreemFS volume that stores the job checkpoint image of a given checkpoint version number. Based on the meta-data process group id and type the checkpointer is instructed to rebuild all structures representing a process group. It can also be indicated to restart from an incremental or full checkpoint. Since rebuilding may vary across grid nodes and intermediate modifications of commonly used resources must be avoided, processes are still stopped.
- *xos_resume_job_unit_rst* The rebuilded processes are waked up and proceed with normal execution. Execution of restart callbacks is enforced.

Furthermore, a generic library provides a common interface for registering user-defined callback functions, see Section 4.3. The callback functions are passed to the checkpointer-specific callback management after a checkpointer has been assigned to a job-unit.

4.2 Process Groups

Different process grouping semantics are used at AEM and kernel checkpointer level. Instead of a process list, kernel checkpointers like BLCR use an identifier to reference a UNIX process group (addressed via PGID), a UNIX session

inria-00346955, version 1 - 12 Dec 2008

inria-00346955, version 1 - 12 Dec 2008

(addressed via SID), a root-process of a process tree. LinuxSSI uses a LinuxSSI application identifier to checkpoint/restart processes. However, AEM uses only the concept of job-unit (addressed by a jobID) to group processes. Since the AEM job checkpointer initiates checkpoint/restart, it merely has a jobID. In general, if process groups used by AEM and kernel checkpointers mismatch, too many or just a subset of the processes are checkpointed/restarted which results in inconsistencies. AEM cannot be hardcoded against one process group type, since various kernel checkpointers use different process groups semantics.

In XtreemGCP this issue is solved by the translation library which ensures, that a kernel checkpointer can reference all processes that AEM captures in a job-unit. In terms of LinuxSSI the translation library manages to put all job-unit processes into an SSI application. The SSI application ID is then determined and can be used for checkpoint/restart. In terms of BLCR the root process is determined based upon the job-unit process list. Since no separate UNIX session or UNIX process group is initiated at job submission, BLCR cannot use these two process group semantics. More information can be found under [18].

Using the root-process of a process tree is dangerous. If an intermediate process fails, the subtree cannot be referenced anymore by the root-process. Linux provides so-called *cgroups* [4] - a mechanism to group tasks and to apply special behaviours onto them. *Cgroups* can be managed from user space via a dedicated file system. Since the upcoming Linux-native checkpointer will be based on *cgroups* and LinuxSSI will be ported towards using *cgroups* - a commmonly used process group semantic on user and kernel checkpointer level has been found which avoids the occurence of mismatchings. *Cgroups* allow all processes to be referenced by one identifier even in case of the application internally establishing multiple UNIX process groups or sessions and in case of intermediate process failures.

Integration of *cgroups* is planned by XtreemOS as soon as the Linux-native checkpointer is available. Management of *cgroups* will also be done within a kernel checkpointer-bound translation library.

4.3 Callbacks

As mentioned before different kernel checkpointers vary in their capability to record and re-establish application states. On the one hand kernel checkpointers limitations, e.g. resources that cannot be checkpointed, must be handled. On the other hand they must be prevented from saving too much to be efficient, e.g. it might not be necessary to save tera-bytes of data-base content with each checkpoint.

Our solution for both aspects is to provide callback functions for application developers, that are executed before and after checkpointing (pre/post-checkpoint callbacks) and after restart. Thus, certain resources can be saved/restored when a checkpointer is not able to. Resources can be saved in an application-related way, unnecessary overhead can be reduced, e.g. large files can be closed during checkpointing and reopened at restart within callbacks. Integration of callbacks into the execution of checkpoint/restart has been proposed and implemented by BLCR, [23]. LinuxSSI has been extended towards this functionality, too.

inria-00346955, version 1 - 12 Dec 2008

To hide kernel checkpointer related semantics of callback management, the common kernel checkpointer API also provides an interface to applications for de/registering callbacks.

4.4 Communication Channels

In order to make consistent distributed snapshots, job-unit dependencies must be addressed, too. The latter occur, if processes of disjunctive job-units exchange messages via communication channels. Saving channel content is required only in terms of coordinated checkpointing for reliable channels. Intransit messages of unreliable channels are considered as they can be lost during fault-free execution.

Most kernel checkpointers do not support checkpointing communication channels or use different approaches. A consistent snapshot can be achieved by each job-unit checkpointer executing a common channel snapshot protocol. This protocol is implemented by XtreemGCP and is integrated into the individual checkpointing sequence for each kernel checkpointer without modifying it.

XtreemGCP drains reliable communication-channels before taking a job-unit snapshot similiar to [23]. Therefore, we intercept *send* and *receive* socket functions. When the checkpointing callback function is executed the draining protocol will be executed by creating a new draining thread after other functions optionally defined by the application programmer have finished. Of course we cannot just stop application threads as messages in transit need to be received first. In order to calculate how many bytes are in transit we count the number of sent and received bytes for each application thread on each grid node. This data can be accessed using the distributed AEM communication infrastructure. The draining thread sets a warning variable causing the interceptor to buffer all outgoing messages of involved threads when calling *send*. All these messages will not be passed to the network. Threads that need to receive in-transit data are allowed to run until they have received all in-transit messages. In the latter case a thread is stopped. Depending on the communication pattern it may take some time to drain all communication channels but by buffering outgoing data the protocol will stop definitively.

During restart all buffered messages need to be injected again. Flushing these buffers is done when restart callback functions are exectuted.

5 Grid Related Issues

5.1 Job Submission

In XtreemOS, the resource requirements of a job are described using the job submission description language (JSDL) [2]. We have extended the JSDL file of a job by checkpointing requirements which are defined in a separate XML file referenced. This extension allows the user/administrator to control checkpointing for specific applications, e.g. the best suited checkpointing strategy can be listed including strategy parameters such as checkpoint frequency or number of checkpoints to be kept. Furthermore, it includes the selection of an appropriate kernel checkpointer that is capable of consistently checkpointing application resources e.g. multi-threaded processes, SYSV IPC objects, etc. that are listed

in the XML file. The discovery of a suitable kernel checkpointer is integrated in the resource discovery component of AEM.

5.2 Checkpoint File Management

A job checkpoint is made up of job-unit checkpoint images and grid meta-data describing the checkpoint. Checkpoint file management must correspond to the checkpointing strategy used in order to maintain consistency and efficiency.

5.2.1 Checkpoint Images and Garbage Collection

Storing checkpoint image files can be costly due to their size and thus resource consumption. Garbage collection (GC) is essential to use disk space efficiently. The job checkpointer implements the GC deleting checkpoint files not needed anymore. When checkpointing is used to migrate or suspend a job, job-unit checkpoint images are removed immediately after the job has been restarted. In these cases, the disk space needed to store the checkpoints is provided by the system.

The system allows the user to define how many checkpoints to keep. However, the user has to provide the necessary disk space. If his quota is exhausted old checkpoints will be deleted. Because of potential creaping errors it is wise to keep at least some older checkpoints, although this is not mandatory within a fail-stop failure model.

Some checkpointing techniques, like incremental checkpointing, may create dependencies between job-units snapshots. In that case, the job-unit checkpointer is in charge of interacting with the kernel checkpointer to identify the checkpoints that can be removed. Furthermore, for uncoordinated checkpointing a set of consistent checkpoints, normally computed at restart, need to be computed during GC in order to determine which checkpoints can be deleted.

5.2.2 Grid-Checkpointing Meta-Data

Meta-data describe job and job-unit related data and include: the kernel checkpointers used, the checkpoint version number, the process list, the process group reference id, the process group reference type (PGID, SID, PID, APPID), etc. The job checkpointer relies on this meta-data during restart, e.g. for reallocating resources valid before checkpointing, using the appropriate kernel checkpointers and for correctly addressing checkpoint images.

Furthermore, grid-checkpointing meta-data can be used for making checkpoint statistics. Adaptive checkpoint policy management relies on such information, see 3.3. Snapshots and grid-checkpointing meta-data can be deleted for disk space efficient usage. However, over a defined time interval selected meta-data information is kept for checkpoint policy management.

5.3 Resource Conflicts

Identifiers of resources such as process IDs, IPC object IDs, etc. being valid before checkpointing must be available after restart to seamlessly proceed with application execution. Failures occur at restart, if these resources are in use by

inria-00346955, version 1 - 12 Dec 2008

other running applications. Currently, these identifiers are saved in the meta-data. Availability of these identifiers on potential restart grid nodes is evaluated at restart by the involved translation library.

However, the best-suited solution is to use light-weight virtualization (e.g. namespaces applied to resources such as process IDs) and resource grouping mechanisms (e.g. cgroups) in order to isolate resources belonging to different applications. An application may use any identifier it needs within a resource group. The kernel is still able to distinguish equally named identifiers used across multiple applications, by isolating resource groups and mapping each identifier to a unique one at kernel level. Integration of state-of-the-art light-weight virtualization mechanisms provided by mainline Linux into XtreemOS is in progress, see also [18].

5.4 Security

Only authorized users should be able to checkpoint and restart a job. XtreemGCP relies on the XtreemOS security services to authenticate and authorize users [1].

Checkpoint/restart can introduce some issues regarding security because things may have changed between checkpoint and restart. For example, a library that was used by the job, and that was possibly checkpointed with the job, may have been updated to correct a security hole. The job owner may have lost some capabilities and is not allowed anymore to access some resources the job was using before restart, e.g. files. We cannot detail all possible cases here, but the general rule XtreemGCP must follow is that transparency comes after security. This means that a job can be restarted only if it does not imply any security rules violation.

6 Related Work

Only a few projects aimed at providing checkpointing for grids. The CoreGRID grid checkpointing architecture (GCA) [15] proposes a solution to enable the use of low-level checkpointers, i.e. kernel, user, or application-level checkpointers. GCA is based on a centralized architecture. The *Grid Checkpoint Service* is the central component of the service that manages checkpoints in the grid and interacts with other grid services. A *Checkpoint Translation Service* executed on each node exposes to the grid level a uniform interface independent from the underlying low-level checkpointer. This interface has not been described in literature in more details. Furthermore, XtreemGCP supports also jobs spanning multiple grid nodes. Another aspect is that CGA prefers the use of Virtual Machines (VMs) instead of kernel checkpointers. Although VMs help to avoid resource conflicts at restart there is still a need to coordinate checkpointing for jobs running on multiple grid nodes. Overall it remains open to which extent CGA supports different kernel checkpointers in their implementation. Another difference is, that GCA does currently not support checkpointing distributed applications.

The Open Grid Forum GridCPR Working Group targets application-level checkpointing [24, 3]. They listed use-cases and defined requirements regarding APIs and related services for application-level checkpointing in grids. A service architecture comparable to the XtreemGCP architecture was deduced. They

inria-00346955, version 1 - 12 Dec 2008

inria-00346955, version 1 - 12 Dec 2008

have also specified a generic API to enable applications or administrative tools to interact with the GridCPR system. Part of this work has been included in SAGA [10]. This work is hard to compare with our common kernel checkpointer API since it is limited to application-level checkpointing. But we plan to synchronize our common kernel checkpointer API with the API designed by OGF.

The hpc4u project checkpointer [9] is based on MCR, a lightweight virtualization solution that avoids resource conflicts at restart. They are able to checkpoint sequential applications, and MPI applications using a dedicated MPI library. The checkpointer is able to checkpoint applications running the same administrative domain, only.

Our work on callbacks and channel flushing mechanisms has been inspired by previous work done by LAM/MPI [23]. The Open MPI Checkpoint/Restart framework [13] is very similar to XtreemGCP but limited to the MPI context. The framework architecture is based on a central coordinator, is able to address various process checkpointers. The user can register callback functions to be executed at checkpoint or restart. Both systems are, by definition, limited to MPI applications. Moreover, since they do not especially target grids, they do not deal with grid specific issues.

Adaptive checkpointing is important within dynamic grids and has been evaluted for parallel processing systems over peer-to-peer networks [19]. The optimized checkpoint interval is determined automatically by estimating network conditions via statistical data collected online like done by XtreemGCP. The Migol service framework [17] replicates checkpoint files and an adaptive strategy is used to select replication targets. Fortunately, we can store checkpoint files in XtreemFS providing fault tolerance by replicating files.

7 Conclusion and Future Work

A grid checkpointing service is useful for implementing job migration and fault tolerance in grids. In both cases the state of a job must be saved and restored on other grid nodes. In this paper we have described the design and implementation of the XtreemGCP service that integrates existing kernel checkpointers. By introducing a common kernel checkpointer API, we are able to access different existing checkpointers in a uniform way.

The proposed architecture is open to support different checkpointing strategies that may be adapted according to evolving failure situations or changing application behaviour. Because of the integrated architecture design there are synergies for both, XtreemOS and XtreemGCP. The OS can use XtreemGCP for load balancing and providing fault tolerance for system services. XtreemGCP can rely on important services like for example security facilities and XtreemFS.

In this paper we have presented solutions for different challenges related to the implementation of the common kernel checkpointer API. We discussed how to bridge different process group management techniques and how to save communication channels in a heteregenous setup. Furthermore, we have introduced a callback mechanism used to execute system-related checkpointing tasks, e.g. used for communication channel draining, but also to provide a gateway for programmer-assisted optimizations, e.g. controlling checkpointing of very large

files. Other grid-related issues, e.g. checkpoint file management and resource conflicts have been discussed, too.

The current prototype supports checkpointing and restarting jobs using BCLR and Kerrighed kernel checkpointers. XtreemOS is an open source project and the XtreemGCP service will be publicly released soon. Although the concepts and ideas described in this paper are presented in the XtreemOS context they can be adapted to any other grid system.

Future work includes implementation of adaptive checkpointing mechanisms, uncoordinated checkpointing strategies, and integration of the announced kernel checkpointer for the Linux mainstream.

References

[1] http://www.xtreemos.eu.

[2] A. Anjomshoaa, F. Brisard, M. Dresher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification, V. 1.0. Technical report, Open Grid Forum, 2005.

[3] R. Badia, R. Hood, T. Kielmann, A. Merzky, C. Morin, S. Pickles, M. Sgaravatto, P. Stodghill, N. Stone, and H. Yeom. Use-Cases and Requirements for Grid Checkpoint and Recovery. Technical Report GFD-I.92, Open Grid Forum, 2004.

[4] S. Bhattiprolu, Eric W. Biederman, S. Hallyn, and D. Lezcano. Virtual servers and checkpoint/restart in mainstream linux. *SIGOPS Operting Systems Review*, 42(5):104–113, 2008.

[5] J. Corbalán, G. Pipan, and T. Cortés. Design of the Architecure for Application Execution Management in XtreemOS. Technical Report D3.3.1, XtreemOS, 2007.

[6] C. Coti, T. Herault, P. Lemarinier, L. Pilard, A. Rezmerita, E. Rodriguez, and F. Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant mpi. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, USA, 2006.

[7] I. Foster and C. Kesselman. The globus project: a status report. *Future Generation Computer Systems*, 15(5-6):607–621, 1999.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15:200–222, 2001.

[9] H. Kohmann G. Schneider and H. Bugge. Fault Tolerant Checkpointing Solution for Clusters and Grid Systems. Technical report, HPC4U project, 2007.

[10] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. SAGA: A Simple API for Grid Applications – High-Level ApplicationProgramming on the Grid. *Computational Methods in Science and Technology*, 12(1), May 2006.

inria-00346955, version 1 - 12 Dec 2008

inria-00346955, version 1 - 12 Dec 2008

[11] A. S. Grimshaw, Wm. A. Wulf, and CORPORATE The Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, 1997.

[12] F. Hupfeld, T. Cortes, B. Kolbeck, E. Focht, M. Hess, J. Malo, J. Marti, J. Stender, and E. Cesario. Xtreemfs - a case for object-based file systems in grids. *Concurrency and Computation: Practice and Experience*, 20(8), 2008.

[13] J. Hursey, J. M. Squyres, T. I. Mattox, and A. Lumsdaine. The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI. In *Proc. of the 21st IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, 2007.

[14] J. Duell, P. Hargrove, and E. Roman. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Technical Report LBNL-54941, Berkeley Lab Technical Report, 2003.

[15] G. Jankowski, R. Januszewski, R. Mikolajczak, M. Stroinski, J. Kovacs, and A. Kertesz. Grid checkpointing architecture - integration of low-level checkpointing capabilites with grid. Technical Report TR-0036, CoreGRID, May 22, 2007.

[16] M. Litzkow and M. Solomon. The evolution of condor checkpointing. *Mobility: processes, computers, and agents*, pages 163–164, 1999.

[17] A. Luckow and B. Schnor. Adaptive checkpoint replication for supporting the fault tolerance of applications in the grid. In *The 7th IEEE International Symposium on Networking Computing and Applications (NCS)*, Cambridge, MA, USA, 2008.

[18] J. Mehnert-Spahn, M. Schoettner, and C. Morin. Checkpoint process groups in a grid environment. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Dunedin, New Zealand, December 2008.

[19] L. Ni and A. Harwood. An adaptive checkpointing scheme for peer-to-peer based volunteer computing work flows. *ArXiv e-prints*, November 2007.

[20] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent Checkpointing under Unix. In *Proceedings of USENIX Winter 1995 Technical Conference*, pages 213–224, New Orleans, Louisiana/U.S.A., January 1995.

[21] H. P. Reiser, R. Kapitza, J. Domaschka, and F. J. Hauck. Fault-tolerant replication based on fragmented objects. In *DAIS 2006: 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, Bologna, Italy, 2006.

[22] M. Romberg. The unicore grid infrastructure. *Scientific Programming*, 10(2):149–157, 2002.

[23] S. Sankaran, J. M. Squyres, B. Barrett, V. Sahay, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing. *International Journal of High Performance Computing Applications*, 19(4):479–493, 2005.

inria-00346955, version 1 - 12 Dec 2008

[24] N. Stone, D. Simmel, T. Kielmann, and A. Merzky. An Architecture for Grid Checkpoint and Recovery Services. Technical Report GFD-I.93, Open Grid Forum, 2007.

[25] G. Vallée, R. Lottiaux, L. Rilling, J-Y. Berthou, I. Dutka-Malhen, and C. Morin. A Case for Single System Image Cluster Operating Systems: the Kerrighed Approach. *Parallel Processing Letters*, 13(2), June 2003.

inria-00346955, version 1 - 12 Dec 2008

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 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